НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки

Обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Онлайн бібліотека»

Виконав:
студент IV курсу, групи ІО-63
Стельмах Андрій Олегович _____

Керівник:
Доцент, с.н.с., к.т.н.,
Антонюк Андрій Іванович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.,
Сімоненко Валерій Павлович _____

Рецензент:
Доцент кафедри ТК, к.т.н.,
Пасько Віктор Петрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет Інформатики та обчислювальної техніки

Обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп’ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Стельмах Андрій Олегович

1. Тема проєкту «Онлайн бібліотека», керівник проєкту Антонюк Андрій Іванович, с.н.с., к.т.н, затверджені наказом по університету від 07 травня 2020р. №1081-с

2. Термін подання студентом проєкту 20 травня 2020р.

3. Вихідні дані до проєкту технічне завдання, теоретичні данні

4. Зміст пояснювальної записки опис предметної області, дослідження сучасних веб сайтів, опис програмного продукту

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

Структурна схема системи, принципова схема, блок-схема алгоритму.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормо контроль	Сімоненко В.П. доктор технічних наук		

7. Дата видачі завдання 20.02.2020 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Затвердження теми роботи	15.11.2019	
2.	Вивчення та аналіз завдання	23.02.2020	
3.	Розробка архітектури та загальної структури систем	10.04.2020	
4.	Розробка структур окремих підсистем	22.04.2020	
5.	Програмна реалізація системи	28.04.2020	
6.	Оформлення пояснювальної записки	04.05.2020	
7.	Захист програмного продукту	14.05.2020	
8.	Передзахист	26.05.2020	
9.	Захист	20.06.2020	

Студент Андрій СТЕЛЬМАХ

Керівник Андрій АНТОНЮК

Анотація

Пояснювальна записка дипломного проєкту складається з трьох розділів, містить 2 таблиці, 3 додатки та 13 джерел – загалом 55 сторінки.

Об’єкт дослідження: створення веб сайту на прикладі онлайн бібліотеки.

Мета дипломного проєкту: обрання найкращого стеку технологій для розробки сучасного веб-сайту, вирішивши основні проблеми сучасних сайтів.

У першому розділі було розглянуто найпопулярніші стеки технологій, проблематику та історію сучасних сайтів.

У другому розділі описано предметну область та визначено кращий стек технологій. Розроблено структуру модулів сайту та його алгоритми роботи.

У третьому розділі реалізовано програмний додаток. Проведено тестування програмного забезпечення.

Ключові слова: Spring, Solr, Java EE, Relation DB.

Annotation

The explanatory note of the diploma project consists of three sections, contains 2 tables, 3 annexes and 13 sources - a total of 55 pages.

The object of study: creating a website on the example of an online library.

The aim of the diploma project:choosing the best technology stack for developing a modern website, solving the main problems of modern websites.

The first section discusses the most popular technology stacks, issues and history of modern sites.

The second section describes the subject area and identifies the best technology stack. The structure of site modules and its algorithms are developed.

In the third section the software application is implemented. Software testing was performed.

Keywords: Spring, Solr, Java EE, Relation DB.

Опис альбому
дипломного проєкту

на тему: «Онлайн бібліотека»

Київ – 2020

№ екз.	Формат	Позначення	Найменування	К. арк.	№ екз.	Прим.
1						
2						
3	А4		*Завдання на дипломний проєкт*			
4						
5	А4	ІАЛЦ.006328.001 ВП	*Відомість дипломного проєкту*	1		
6						
7	А4	ІАЛЦ.006328.002 ТЗ	*Технічне завдання*	2		
8						
9	А4	ІАЛЦ.006328.003 ПЗ	*Пояснювальна записка*	55		
10						
11	А3	ІАЛЦ.006328.004 Д1	*Алгоритм відображення*	1		
12			*книги*			
13			*Схема принципова*			
14						
15	А3	ІАЛЦ.006328.005 Д2	*Схема зв'язків класів*	1		
16			*Схема функціональна*			
17						
18	А3	ІАЛЦ.006328.006 Д3	*Клієнт серверна взаємодія*	1		
19			Схема структурна			
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						
33						
34						
35						
36						

Змн.	Арк.	№ докум.	Підпис	Дата	*ІАЛЦ.006328.001 ВП*		
Виконав		*Стельмах А.О*			Відомість дипломного проєкту	Літ.	Арк. 1 / Аркушів 1
Керівник		*Антонюк А.І.*					
Н. Контр.		*Сімоненко В.П.*				*НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ гр. ІО-63*	
Зав.каф.		*Стіренко С.Г.*					

Технічне завдання
до дипломного проєкту

на тему: «Онлайн бібліотека»

Київ – 2020

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ 2
2. ПІДСТАВИ ДЛЯ РОЗРОБКИ 2
3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ 2
4. ДЖЕРЕЛА РОЗРОБКИ 2
5. ТЕХНІЧНІ ВИМОГИ 2
5.1. Вимоги до розроблюваного продукту..... 2
5.2. Вимоги до програмного забезпечення..... 3
5.3. Вимоги до апаратної частини 3
6. ЕТАПИ РОЗРОБКИ 3

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку онлайн бібліотеки. Область застосування: розробка серверної частини веб-додатків.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України
«Київський Політехнічний інститут» Ім. Ігоря Сікорського.

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка онлайн бібліотеки за допомогою сучасного стеку технологій.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, а також публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Можливість вільного додавання нових модулів;
- Можливість повнотекстового пошуку;
- Тестування розробленого додатку.

5.2. Вимоги до програмного забезпечення

- Операційна система Linux, macOS, Windows 7, Windows XP або новіше;
- Java 8 або новіше;

5.3. Вимоги до апаратної частини

- Комп’ютер на базі AMD Athlon 200GE або новіше;
- Оперативної пам’яті не менше 256 Мбайт;
- Вільний простір жорсткого диску не менше 200 Мбайт;
- Підключення до інтернету;

6. ЕТАПИ РОЗРОБКИ

	Дата
Збір матеріалу	22.03.2020
Вивчення матеріалу	04.04.2020
Розробка технічного завдання	14.04.2020
Аналіз структури програмного проєкту	23.04.2020
Створення модулів системи	03.05.2020
Тестування окремих модулів системи	11.05.2020
Оформлення документації дипломної роботи	22.05.2020

Пояснювальна записка
до дипломного проєкту
на тему: «Онлайн бібліотека»

Київ – 2020 року

ЗМІСТ

РОЗДІЛ 1. ОГЛЯД ТЕХНОЛОГІЙ ТА ГОТОВИХ РІШЕНЬ 5
1.1 Relational Database 5
1.2 SOLR 5
1.2.1. Особливості Apache Solr 7
1.2.2. Lucene 8
1.3 JAVA EE 9
1.3.1. Історія розвитку JAVA EE 11
1.4. Spring 11
1.4.1 Основні терміни 11
1.4.2. Модульність Spring 12
1.4.3. Spring + Sql (Spring Data JDBC) 15
1.4.4. Spring Data Solr 17
ВИСНОВКИ ДО РОЗДІЛУ 1 21
РОЗДІЛ 2. ПРОЕКТУВАННЯ МОДУЛЯ 22
2.1 SOLR vs Relational Database 22
2.2. JAVA EE VS SPRING 24
2.2.2. Стек Spring 26
2.2.3. Попарне порівняння стеків 28
2.3. Тестовий модуль 30
2.4. Розробка алгоритму 33
ВИСНОВКИ ДО РОЗДІЛУ 2 36
РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ 37
3.1 Огляд призначення розроблюваної системи 37
3.2 Огляд технологічної бази 38
3.2.1 Java 38
3.2.2 Spring 38
3.2.4 JS CSS HTML 40
3.2.5 Thymeleaf 40

3.3. Бізнес-кейси 41
3.3.1. Failed Login 41
3.3.2. Успішний Login 42
3.3.4 Реєстрація 44
3.4. Тестування 46
3.4.1. Терміни 46
3.4.2. Типи тестування 46
3.5. Приклади тестування системи 47
3.5.1. Create 47
3.5.2. Update 47
3.5.3. Delete 48
3.5.4. Клас для тестування дії з базою даних 49
3.5.5. Метод для тестування реєстрації користувача 50
3.5.6. Підсумок тестування 51
ВИСНОВКИ ДО РОЗДІЛУ 3 53
ВИСНОВКИ 54
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 55

ВСТУП

Перший у світі сайт з'явився у 1990 автором якого був Тім Бернерс-Лі. На своєму сайті він описав нову технологію WWW. З цього моменту й почали свою історію веб-сайти. Перші сайти мали вигляд звичайних сторінок з текстом, інколи з посиланням. Зараз же навіть найпростіші сайти мають безліч модулів та складний функціонал. З розвитком веб технологій у сайтів виникли дві великі проблеми. Це потреба модульності та в наслідок потреби зберіганні великої кількість даних проблема проходження пошук за цими даними. Навіть найпростіші сучасні сайти зараз мають складну базу даних з десятками таблиць, сотнями стовпців та сотнями тисяч даних. Зрозуміло що пошук в такій кількості даних може викликати певні проблеми. Також будь який хороших сайт повинен розвиватися, і з часом виникає потреба в додаванні нових окремих модулів.

У даній роботі будуть розглянуті найпопулярніші стеки технологій для створенння сайтів. Буде проведено аналіз цих стеків та їх порівняння. Для порівняння було обрано стек Java EE та сімейство Spring фреймворків. Також буде проведено порівняння між перевагами та недоліками реляційних баз даних та Apache Solr.

У результаті цих порівнянь та на основі отриманих результатів буде створено сайт онлайн бібліотеки за допомогою кращих стеків.

РОЗДІЛ 1. ОГЛЯД ТЕХНОЛОГІЙ ТА ГОТОВИХ РІШЕНЬ

1.1 Relational Database

Реляційна база даних - це набір даних із попередньо визначеними зв'язками між ними. Ці дані організовані у вигляді набору таблиць, що складаються з стовпців та рядків. У таблицях зберігається інформація про об'єкти, представлені в базах даних. У кожному стовпцю таблиці представляє визначений тип даних, у кожній комірці - значення атрибута. Кожен рядок таблиці представляє собою набір пов'язаних значень, що стосуються одного предмета або сутності. Кожна строка в таблиці може бути помічена унікальним ідентифікатором, який називають первинним ключовим словом, а рядки з декількох таблиць можуть бути пов'язані за допомогою зовнішніх ключів. До таких даних можна отримати доступні багатьма способами, і при цьому реорганізовувати таблицю БД не потрібно.

Основна перевага підходу реляційної бази даних - можливість створювати змістовну інформацію, приєднуючись до таблиць. Приєднання таблиць дозволяє зрозуміти зв’язки між даними або способи з'єднання таблиць. SQL включає можливість рахувати, додавати, групувати, а також комбінувати запити. SQL може виконувати основні математичні та субтотальні функції та логічні перетворення. Користувач може задати запит за датою, назвою чи будь-яким стовпцем. Ці особливості роблять реляційний підхід найпопулярнішим інструментом запитів у бізнесі сьогодні.

1.2 SOLR

Solr - це платформа для пошуку з відкритим кодом, написана на Java, від проекту Apache Lucene. Основні його функції включають повнотекстовий пошук, виділення звернень, граничний пошук, індексацію в режимі реального часу, динамічну кластеризацію, інтеграцію баз даних, функції NoSQL [2] та обробку документів (наприклад, Word, PDF). Забезпечуючи розподілену пошук та реплікацію індексу, Solr розроблений для масштабованості та відмовостійкості. [3] Solr широко застосовується для повнотекстового пошуку та індексації, має активну спільноту розробки та регулярні оновлення.

Solr працює як окремий сервер повнотекстового пошуку. Він використовує бібліотеку пошуку Lucene Java в своїй основі для повнотекстової індексації та пошуку, а також має REST-подібні HTTP / XML та JSON API, які роблять її зручною для використання у найпопулярніших мовах програмування. Зовнішня конфігурація Solr дозволяє адаптувати її до багатьох типів додатків без кодування Java, і вона має архітектуру плагінів для підтримки більш досконалої настройки.

Apache Lucene та Apache Solr виробляються однією командою розробників Apache Software Foundation.

Solr був побудований поверх Lucene (повнотекстова пошукова система). Solr готовий до роботи, швидкий і масштабується. Додатки, створені з використанням Solr, є складними і забезпечують високу продуктивність.

У 2004 році Йоник Сілі створив Solr за потреби пошуку на веб-сайт компанії CNET Networks. У січні 2006 року був зроблений проект з відкритим вихідним кодом в рамках Apache Software Foundation. Його остання версія, Solr 6.0, була випущена в 2016 році з підтримкою виконання паралельних SQL-запитів.

Solr можна використовувати разом з Hadoop. Оскільки Hadoop обробляє великий обсяг даних, Solr допомагає нам знайти необхідну інформацію з такого великого джерела. Solr може використовуватися не тільки для пошуку, але і для зберігання. Як і інші бази даних NoSQL, це нереляційних технологія зберігання і обробки даних.

Коротше кажучи, Solr - це масштабована, готова до розгортання система пошуку і зберігання, оптимізована для пошуку великих обсягів тексто-орієнтованих даних.

1.2.1. Особливості Apache Solr

Solr - це оболонка навколо Java API від Lucene. Тому, використовуючи Solr, ви можете використовувати всі можливості Lucene. Давайте подивимося на деякі з найбільш важливих особливостей Solr -

- **Restful APIs** –Щоб спілкуватися з Solr, не обов'язково мати навички програмування на Java. Замість цього ви можете використовувати інтерфейс для спілкування з ним. Ми вводимо документи в Solr в таких форматах, як XML, JSON і .CSV, і отримуємо результати в тих же форматах.
- **Повнотекстовий пошук** – Solr надає всі можливості, необхідні для повнотекстового пошуку, такі як токени, фрази, перевірка орфографії, символи узагальнення і автозаповнення.
- **Готовність** –в залежності від потреб організації Solr може бути розгорнуто в будь-яких системах (великих чи малих), таких як автономні, розподілені, хмарні і т. Д.
- **Гнучкість та можливість розширення.** Розширюючи класи Java і налаштовуючи їх відповідним чином, ми можемо легко налаштовувати компоненти Solr.
- **База даних NoSQL** – Solr також може використовуватися в якості бази даних NOSQL з великими обсягами даних, де ми можемо розподіляти завдання пошуку по кластеру.
- **Інтерфейс адміністратора** –Solr надає простий у використанні, зручний, функціональний користувальницький інтерфейс, за допомогою якого ми можемо виконувати всі можливі завдання, такі як управління журналами, додавання, видалення, оновлення і пошук документів.
- **Висока масштабованість .**Використовуючи Solr з Hadoop, ми можемо масштабувати його ємність, додаючи репліки.

- **Текст-ориентированный и отсортированный по релевантности** – Solr в основному використовується для пошуку текстових документів, а результати надаються відповідно до запиту користувача по порядку.

На відміну від Lucene, вам не потрібно мати навички програмування на Java при роботі з Apache Solr. Він надає чудову готову до розгортання службу для створення вікна пошуку з автозаповненням, яке Lucene не надає. Використовуючи Solr, ми можемо масштабувати, поширювати і управляти індексами для великомасштабних додатків (великих даних).

1.2.2. Lucene

Lucene - це проста, але потужна бібліотека пошуку на основі Java. Його можна використовувати в будь-якому додатку для додавання можливостей пошуку. Lucene - це масштабована і високопродуктивна бібліотека, яка використовується для індексації і пошуку практично будь-якого тексту. Бібліотека Lucene надає основні операції, необхідні для будь-якого пошукового додатка, такі як індексування і пошук.

Якщо у нас є веб-портал з величезним обсягом даних, то нам, швидше за все, буде потрібно пошукова система на нашому порталі для отримання відповідної інформації з величезного пулу даних. Lucene працює як серце будь-якого пошукового додатка і забезпечує життєво важливі операції, пов'язані з індексацією і пошуком.

1.2.3. Apache Solr – Основи пошукової системи

Пошукова система відноситься до величезної бази даних інтернет-ресурсів, таких як веб-сторінки, групи новин, програми, зображення і т. Д. Вона допомагає знаходити інформацію в World Wide Web.Користувачі можуть шукати інформацію, передаючи запити в пошукову систему у вигляді ключових слів або фраз. Потім пошукова система виконує пошук в своїй базі даних і повертає відповідні посилання користувачеві.

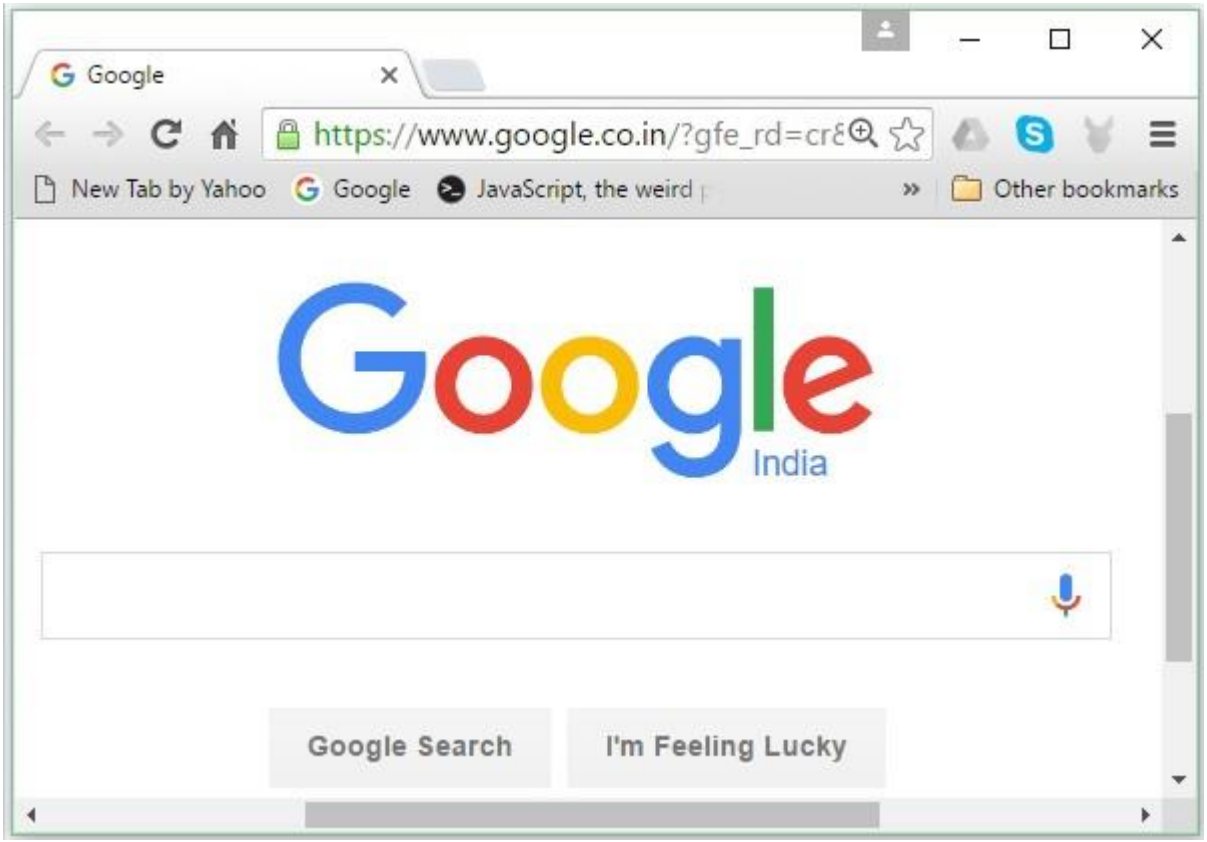


Рисунок 1.1. Приклад пошукової системи

Компоненти пошукової системи

Як правило, є три основних компоненти пошукової системи, як зазначено нижче -

- **Web Crawler** – веб-сканери також відомі як павуки або боти. Це програмний компонент, який проходить через Інтернет для збору інформації.
- **База данных** –Вся інформація в Інтернеті зберігається в базах даних. Вони містять величезний обсяг веб-ресурсів.
- **Интерфейсы поиска** – цей компонент є інтерфейсом між користувачем і базою даних. Це допомагає користувачеві здійснювати пошук по базі даних.

1.3 JAVA EE

Java Platform, Enterprise Edition — платформа Java яка надає API та середовище для розробки програмного забезпечення. Java EE розширює стандартну платформу Java SE - Java Standart Edition.

J2EE в основному використовують у маштабних, високопродуктивних проектах, у яких необхідна надійність, масштабованість, гнучкість. Компанія Oracle, яка придбала Sun, активно просуває Java EE у поєднанні зі своїми технологіями, зокрема з СУБД Oracle.

Java EE або Java Enterprise Edition являє платформу для створення корпоративних додатків на мові Java. Перш за все це сфера веб-додатків і веб-сервісів. Java EE складається з набору API і середовища виконання.

Деякі з API:

- **Enterprise JavaBeans (EJB)** представляють класи, які зберігають бізнес-логіку.
- **JavaServer Pages (JSP).** Також модулі на стороні сервера, які обробляють запити. Зручні для генерації великої контенту HTML. По суті являє собою сторінки з кодом HTML / JavaScript / CSS з вкрапленнями коду на Java
- **Java Servlets.** Сервлети представляють спеціальні модулі, які обробляють запити від користувачів і відправляють результат обробки.
- **JSON Processing (JSON-P)** дозволяє працювати з рядками JSON в Java
- **Contexts and Dependency Injection (CDI)** надає механізм для впровадження та управління залежностями в інші об'єкти.
- **WebSocket** дозволяє інтегрувати WebSocket в додатки на Java.
- **JSON Binding (JSON-B)** надає функціонал для сериализации і десеріалізациі JSON в об'єкти Java.
- **Java Message Service (JMS)** - API для пересилання повідомлень між двома і Java API for RESTful Web Services (JAX-RS) - API для застосування архітектури REST в додатках.
- **JavaServer Faces (JSF)** надає можливості для створення призначеного для користувача інтерфейсу на стороні сервера.
- **Security API** - API для стандартизації та спрощення завдань забезпечення безпеки в додатках на Java.

Всі ці API, та ще безліч інших об’єднуються в групу, що називається Java EE. Варто відзначити, що також в середовищі веб-розробки на Java популярна ще одна технологія Spring . Spring не є частиною Java EE це більше як альтернативний підхід до створення веб-додатків на мові Java, але про нього ми поговоримо пізніше.

1.3.1. Історія розвитку JAVA EE

Предтечею Java EE був проект JPE Project, який стартував у травні 1998 року. У грудні 1999 року вийшов реліз платформи J2EE 1.2(Enterprise Java Platform), яка об'єднувала такі компоненти як сервлети, Java Server Pages, Enterprise JavaBeans та Java Message Service . У 2006 році з виходом 5-й версії вона була перейменована в JEE(Java Enterprise Edition). З тих пір періодично виходять нові версії платформи. Остання поточна версія - Java EE 8 вийшла у вересні 2017 року.

У 2017 році відбулася нова віха у розвитку платформи, а саме компанія Oracle передала управління над розвитком Java EE організації Eclipse Foundation. І у квітні 2018 року Java EE була перейменована в Jakarta EE.

1.4. Spring

Spring Framework - це фреймворк(сімейство фреймворків) з відкритим кодом та контейнерами для платформи Java.

Основна частина Spring Framework може використовувати будь-яку версію Java, яка використовується для створення веб-додатків на платформі Java EE. Spring Framework не використовує конкретні параметри програмування, Spring Framework став популярним як альтернатива до JEE.

1.4.1 Основні терміни

Для розуміння Spring для початку нам потрібно ознайомитися з частиною шаблонів та принципів об'єктно орієнтованого програмування які він реалізує.

Inversion of Control (інверсія управління) - це якийсь абстрактний принцип, набір рекомендацій для написання слабо пов'язаного коду. Суть якого в тому, що кожен компонент системи повинен бути якомога більш ізольованим

від інших, не покладаючись в своїй роботі на деталі конкретної реалізації інших компонентів.

Dependency Injection (впровадження залежностей) - це одна з реалізацій IoC (крім цього є ще Factory Method, Service Locator).

IoC-контейнер - це бібліотека, яка дозволить вам спростити і автоматизувати написання коду з використанням даного підходу настільки, наскільки це можливо.

Паттерн Factory Method(Фабричний метод)

Factory Method — це шаблон проектування породжуючого типу , який використовують при проблемі створення різних об'єктів, без прив’язки коду до самих класів цих об'єктів.

Factory Method створює метод, який використовують замість створення об'єктів за допомогою конструкторів. Наслідники можуть перевизначити цей метод, за необхідності зміни типу створюваних продуктів.

1.4.2. Модульність Spring

Як було сказано вище, проблема більшості сайті в тому що вони розвиваються і виникає потреба в модульності. Spring Framework найкраще підходить для вирішення цієї проблеми, оскільки він складається з декількох модулів, які надають широкий спектр послуг:

- **Dependency Injection**: конфігурація компонентів додатків та управління життєвим циклом об'єктів Java здійснюється головним чином за допомогою управління інверсією.
- **Aspect-oriented programming** , яке дає можливість реалізовувати цілісні процедури
- **Data Access/Integration**: дає можливість для роботи з реляційною системою управління БД з використанням JDBC та об'єктно-реляційних відображень, також дає доступ та інструменти для роботи з базами даних NoSQL
- **Model-View-Controller**: Програмне забезпечення на базі сервлету HTTP, що надає RESTful веб-додатки та веб-сервіси.

- **Authentication and authorization**: настроювані процеси захисту, що підтримують різноманітні стандарти, протоколи, інструменти та практики через підпроект Spring Security.
- **IoC, Inversion of Control**: конфігурації для управління об'єктами Java для локальної або віддаленої конфігурації через JMX
- **Testing**: зручна база для тестування за допомогою JUNIT або інтеграційними тестами.

Розглянемо детальніше основні модулі, що б краще зрозуміти в чому їхні переваги, та чому це рішення підходить для вирішення нашої проблеми.

1.4.2.1. Основний контейнер(Core Container)

Базовий контейнер складається з модулів Core, Beans, Context і Expression Language, подробиці яких наступні:

Модуль Core забезпечує основні частини платформи, включаючи функції IoC і Dependency Injection.

Модуль Bean надає BeanFactory, яка представляє собою складну реалізацію фабричного шаблону.

Модуль Context заснований на міцній основі, що надається модулями Core і Beans, і є середовищем для доступу до будь-яких об'єктів, певним і налаштованим. Інтерфейс ApplicationContext є координаційним центром модуля Context.

Модуль SpEL надає потужний мову виразів для запитів і маніпулювання графом об'єктів під час виконання.

1.4.2.2. Доступ до даних / інтеграція

Рівень доступу до даних / інтеграції складається з модулів JDBC, ORM, OXM, JMS і Transaction, подробиці яких наступні:

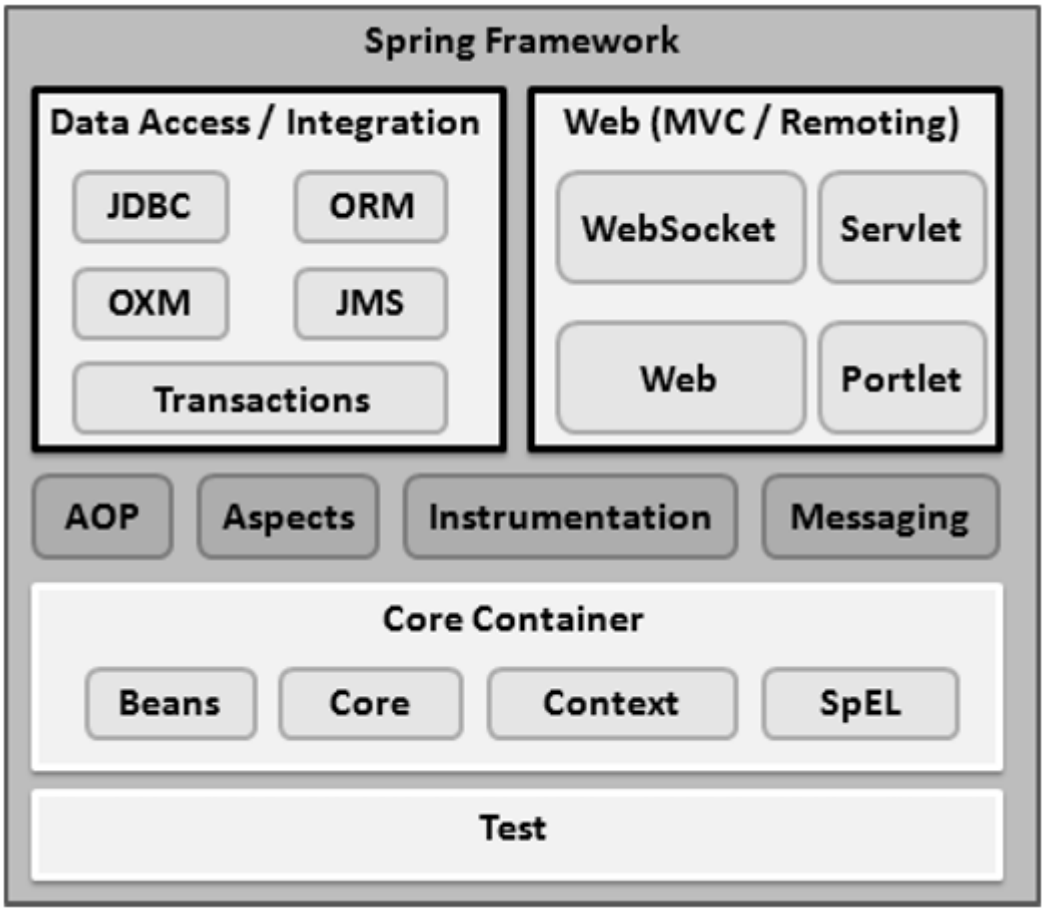


Рисунок 1.3. Основні модулі Spring фреймворка

- **Модуль JDBC** надає рівень абстракції JDBC, який усуває необхідність в осоружному кодуванні, пов'язаному з JDBC.
- **Модуль ORM** надає шари інтеграції для популярних API об'єктно-реляційного відображення, включаючи JPA, JDO, Hibernate і iBatis.
- **Модуль OXM** надає рівень абстракції, який підтримує реалізації відображення об'єктів / XML для JAXB, Castor, XMLBeans, JiBX і XStream.
- **Модуль JMS Java Messaging Service** містить функції для створення і споживання повідомлень.
- **Модуль Transaction** підтримує програмне і декларативне управління транзакціями для класів, які реалізують спеціальні інтерфейси, і для всіх ваших POJO.

1.4.2.3. Web

Веб-шар складається з модулів Web, Web-MVC, Web-Socket і Web-Portlet, подробиці яких наведені нижче:

- **Веб-модуль** забезпечує базові функції веб-інтеграції, такі як функція багатоетапної завантаження файлів і ініціалізація контейнера IoC з використанням Прослуховувач сервлетів і контексту веб-орієнтованого додатки.
- **Модуль Web-MVC** містить реалізацію Spring-Model-View-Controller (MVC) для веб-додатків.
- **Модуль WebSocket** забезпечує підтримку двостороннього зв'язку на основі WebSocket між клієнтом і сервером в веб-додатках.
- **Модуль Web- портлету** надає реалізацію MVC для використання в середовищі портлету і відображає функціональність модуля Web-Servlet.

1.4.3. Spring + Sql (Spring Data JDBC)

Ідея Spring Data JDBC полягає в тому, щоб надати доступ до реляційних баз даних без використання всієї складності JPA.

JPA пропонує такі функції, як лінива завантаження (lazy loading), кешування і відстеження змін (dirty tracking). Не дивлячись на те, що ці фічі дуже круті, якщо вони, звичайно, вам дійсно потрібні, вони можуть сильно ускладнити розуміння логіки доступу до даних.

Механізм ледачою завантаження може раптово виконати ресурсомісткі запити, або і зовсім впасти з виключенням. Кешування може встати на вашому шляху коли ви вирішите порівняти дві версії entity, а вкупі з відстеженням змін завадить зрозуміти - в який же момент реально виконуватися всі операції з базою даних?

Spring Data JDBC фокусується на набагато більш простої моделі. Чи не буде кешування, відстеження змін, або ледачою завантаження. Замість цього, SQL запити будуть виконані тоді і тільки тоді, коли ви викликали метод сховища. Повертається результат буде повністю завантажений в пам'ять після

виконання методу. Чи не буде і механізму "сесії" або проксі-об'єктів для entities. І все це повинно зробити Spring Data JDBC простішим і зрозумілим інструментів для доступу до даних.

Зрозуміло, такий спрощений підхід виливається в цілий ряд обмежень, про які ми поговоримо в наступних постах. Грядущий реліз це найперша версія бібліотеки, у нас є багато планів і задумів, які ми хочемо реалізувати, але ми змушені їх відкласти, щоб дати вам можливість почати користуватися Spring Data JDBC якомога раніше.

Коли нам доводиться працювати з простим JDBC, такі операції, як відкриття / закриття з'єднання з базою даних (далі - БД), обробка виключень і т.д. роблять код вкрай громіздким і складним для читання.

Реалізація JDBC в Spring Framework бере на себе роботу з багатьма низькорівневими операціями (відкриття / закриття з'єднань, виконання SQL-запитів і т.д.).

Завдяки цьому при роботі з БД в Spring Framework, нам необхідно тільки визначити параметри з'єднання з БД і прописати SQL-запит, решту роботи за нас виконує Spring.

JDBC в Spring має кілька класів (кілька підходів) для взаємодії з БД. Найбільш поширені з них - це використання класу JdbcTemplate. Це базовий клас, який управляє обробкою всіх подій і зв'язками з БД.

Клас JdbcTemplate виконує SQL-запити, виконує ітерації по ResultSet і витягує викликаються значення, оновлює інструкції та виклики процедур, "ловить" виключення і транслює їх у виключення, певні в пакеті org.springframwork.dao.

Екземпляри класу JdbcTemplate є потікозахищеними. Це означає, що налаштувавши єдиний екземпляр класу JdbcTemplate, ми можемо потім його використовувати для декількох об'єктів DAO.

При використанні JdbcTemplate, найчастіше, він конфігурується в файлі конфігурації Spring. Після цього він впроваджується допомогою як бін в класи DAO.

1.4.4. Spring Data Solr

У цьому пункту ми розглянемо Spring Data Solr на практиці, та на прикладі нашого BookRepository .

Solr - це відкритий вихідний код, готовий до розгортання корпоративного повнотекстового пошукового движка.

Розглянемо, як виконати прості налаштування Solr і, звичайно, як взаємодіяти з сервером.

По-перше, нам потрібно запустити сервер Solr і створити ядро для зберігання даних (яке за замовчуванням Solr створюватиме в режимі без схеми).

Spring Data

Як і у будь-якого іншого проекту Spring Data, Spring Data Solr має чітку мету - видалити стандартні коди, якими ми обов'язково скористаємося.

Створення документа

Створимо документ під назвою Book:

```
/**
* class Book with properties name, genre, author, description, image, date,rating, countRating;
* @autor STS
* @version 1.1
*/
@Entity
@SolrDocument(solrCoreName = "book")
public class Book  {


  @Id   @GeneratedValue(strategy=GenerationType.IDENTITY)
  @Indexed(name = "id")
  private Long id;

  @Indexed(name = "name")
  private String name;
```

@ SolrDocument анотація вказує, що клас Book є документом Solr і індексується в ядро з ім'ям book. Поля, відмічені @ Indexed, індексуються в Solr і будуть доступні для пошуку.

Визначення інтерфейсу сховища

Далі створюємо інтерфейс сховища, розширивши репозиторій, наданий Spring Data Solr. Природно, ми будемо параметризувати це з Book і String в якості ідентифікатора нашої сутності:

```
@Repository
public interface BookRepository extends JpaRepository<Book, Long> {
  /**
   * method for getting all books from db
   * @return allBooks
   */



  List<Book> findAll();

  /**
   * method for getting all books order by rating
   * @return booksOrderById
   */

  List<Book> findAllByOrderByRatingDesc();

  /**
   * method for getting book by id
   * @param id;
   * @return bookById
   */
  Optional<Book> findById(Long id);
```

Ми використовуємо @ EnableSolrRepositories для сканування пакетів на наявність репозиторіїв. Зверніть увагу, що ми вказали розташування файлу властивостей іменованого запиту і включили підтримку багатоядерності.

Якщо багатоядерний процесор не включений, то за замовчуванням Спрінг буде припускати, що конфігурація Solr призначена для одного ядра.

Індексування, оновлення та видалення

Для пошуку документів в Solr документи повинні бути проіндексовані в сховище Solr.

Наступний приклад індексує документ книги в репозиторії Solr за допомогою методу SolrCrudRepository's save:

```
Book book= new Book ();
book.setId ( "1");
book.setName ( "Arture");
bookRepository.save (book);
```

Тепер знайдемо і оновимо документ:

```
Book retrievedBook = bookRepository.find ( "1");
retrievedBook.setName ( "Harry Potter");
bookRepository.save (retrievedBook);
```

Документи можна видалити, викликавши метод delete:

```
bookRepository.delete (retrievedBook);
```

Генерація запиту на основі імені методу

Запити на основі імені генеруються шляхом аналізу імені методу :

У нашому інтерфейсі сховища є метод findBookByName, який генерує запит на основі імені методу:

ArrayList retrievedBooks = bookRepository.findBookByName ("Book");

Запит з @ Query анотацією Пошукові запити Solr можуть бути створені при наявності запиту в анотації @ Query методу. У нашому прикладі findByNameQuery позначається анотацією @ Query:

```
@Query ( "name: **? 0 **")
public Author
findByNameQuery (String searchTerm);
```

Давайте використовувати цей метод для отримання документів:

Author author = BookRepository.findByNameQuery ("Harry Potter", new PageRequest (0, 10));

Викликаючи findByName, ми отримуємо першу сторінку документів rybu, які містять слово «Harry Potter» в будь-якому з полів id або name.

Іменований запит

Іменовані запити аналогічні запитам з анотацією @ Query, за винятком того, що запити оголошуються в окремому файлі властивостей:

@Query (name = "Book.findByNamedQuery")

public Author

findByName (String searchTerm, Pageable pageable);

Зверніть увагу, що анотація @ Query не потрібно, якщо ключ (findByNamedQuery) запиту у файлі властивостей збігається з ім'ям методу.

ВИСНОВКИ ДО РОЗДІЛУ 1

У ході розробки даного розділу дипломної роботи, було розглянуто проблеми створення сучасних сайтів. Проаналізовано ринок, та обрано найбільш популярні рішення для створення сайтів. Оглянуто їхні характеристики та переваги та недоліки. Рішення були проаналізовані та виділено дві найбільші проблеми. А саме:

- Сучасні сайти з часом вимагають нового функціоналу та можливостей, для цього(за часту) переписується значна частина кода, через що виникають проблемні ділянки, нові помилки, баги та проблеми з продуктивністю.
- Зараз у більшості сайтів існує пошукова система, будь то торговий сайт чи сайт для навчання, розваг майжу у будь якому сайті існує пошук. І у більшості випадків цьому пошуку хочеться побажати тільки кращого. Навіть у популярних та великих сайтах спричинивши найменшу граматичну помилку ми можемо не отримати потрібного результату.

Отже в цій роботі я буду шукати оптимальне рішення та набір інструментів для створення сучасного сайту з можливістю росту, додаванням нових модулів та хорошою, якісною пошуковою системою.

РОЗДІЛ 2. ПРОЕКТУВАННЯ МОДУЛЯ

2.1 SOLR vs Relational Database

Якщо припустити, що користувач має реляційний БД, навіщо використовувати Solr? Якщо у випадку використання ви вимагаєте, щоб людина вводила слова у вікні пошуку, вам потрібна текстова пошукова система на зразок Solr.

Бази даних та Solr мають додаткові сильні та слабкі сторони. SQL підтримує дуже простий текстовий пошук на основі символів підстановки з деякими простими нормалізаціями, такими як відповідність верхнього регістру до нижнього регістру. Проблема полягає в тому, що це повне сканування таблиці. У Solr усі слова, які можна шукати, зберігаються у "зворотному індексі", який швидше здійснює пошук на порядки.

Таблиця 2.1. Порівняння Solr та Relational Database

	Solr	Relational Database
Lucene	Solr	Relational DB
Текстовий пошук	Швидкий та витончений	мінімальний та повільний
Додатки (Features)	Небагато, націлені на пошук тексту	Багато
Складність розгортання	Середня	Низька
Інструменти адміністрування	Мінімальні проекти з відкритим кодом	Багато з відкритим кодом та комерційними
Інструменти моніторингу	Слабкі	Дуже сильні та хорошої якості
Інструменти масштабування	Автоматизований, середнього масштабу	Великі масштаби

Продовження Таблиця 2.1. Порівняння Solr та Relational Database

	Solr	Relational Database
Підтримка	Слабка	Сильна
Гнучкість схеми	Потребує в загальній перебудові	Зміни відразу відображаються
Швидкість індексації	Повільна	Швидше і регульоване
Швидкість запитів	Пошук тексту - швидкий і передбачуваний	Дуже залежить від дизайну та використання
Швидкість додавання / вилучення рядків	Повільна	Швидка
Часткова модифікація запису	Відсутня	Присутня
Час видимості після додавання	Повільний	Помірний
Доступ до внутрішніх структур даних	Високий	Відсутній
Необхідні технічні знання	Java (minimal), web server deployment, IT	SQL, DB-specific factors, IT

З точки зору бази даних, індекс Люсіни можна розглядати як одну таблицю БД з дуже швидкими підборами та цікавими удосконаленнями для пошуку тексту. Цей показник порівняно дорогий у просторі та часі створення. Solr обертає цей API повнофункціональною обгорткою, надаючи ці доповнення:

- функції схеми та функції обробки тексту, які відповідають більшості розгортань Lucene
- чисте розгортання як веб-сервіс для індексації та пошуку

- зручна масштабованість на декількох серверах
- крива навчання та покращення прийняття на ~ 2 порядки

2.2. JAVA EE VS SPRING

Рисунок 2.1. Емблеми двох найпопулярніших стеків

Практично всі великі проекти на Java написані на Java EE. Для більшості великий проект на Java = Java EE, без варіантів. Спроби розповісти людям, що в світі є безліч інших технологій, викликають непідробне здивування в їхніх очах. Ми ж будемо використовувати Spring. Зараз розглянемо “за” та “проти” цього рішення.

2.2.1. Стек Java EE

Почнемо з того, що розглянемо запитання чому ж більшість компаній обирають Java EE ,а не Spring. Розробка на стеку Java EE виявляється істотно дешевше розробки на стеку Spring при деяких обмеженнях. А оскільки для більшості компаній важливі тільки гроші - то ось вам і відповідь. Та як ми розглядаємо проблему не матеріальну а більш технічну то перейдемо до технічних характеристик одного та іншого стеку. Для початку на Рисунку 2.2. зображено приклад схеми роботи простого, невеликого додатку написаного на стеку технологій Java EE.

Як бачимо, нічого складного взагалі. Запит обробляється JSF сторінкою, а потім - біном. Управління далі йде на бізнес-логіку в EJB, яка працює з базою даних через JPA. Поки все просто і зрозуміло.

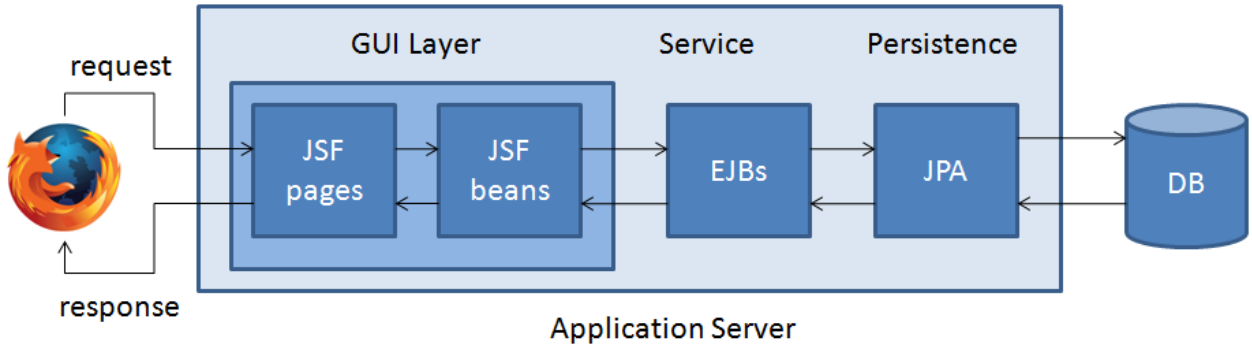


Рисунок 2.2. Спрощена схема роботи Java EE додатку

Найприємніше у всій цій картині, що якщо навантаження на цю програму збільшиться на кілька порядків, то його схема не зміниться взагалі ніяк. Додаток треба буде встановити просто на кластер з декількох потужних нод, і все буде працювати в кластерному оточенні без найменших правок. Якщо потужності все одно буде не вистачати, то досить додати ще нод в кластер - і її вистачить. Це також одна з найбільших технічних причин вибору стеку Java EE. Простота самого механізму роботи. Але це всього лиш примітивний Java EE додаток. Подивимося на щось більш масштабне.

Як показано на Рисунку 2.3. виглядає великий проект на Java EE з точки зору його документації. Бачимо що вже все набагато складніше, додалося безліч нових модулів і вже не кожен здатен зрозуміти цю схему. Причому механізм роботи попередньої схеми і цієї повністю збігається. З різницею в декілька ускладнень, таких як розширення можливостей. Але зовнішня структура однакова. Ми бачимо рівень відображення (GUI Layer), рівень обробки, або сервісний рівень(Service) та рівень роботи з базою даних(Persistens). Але поки що ми це просто запам'ятаємо і рушимо вперед розглядати Spring.

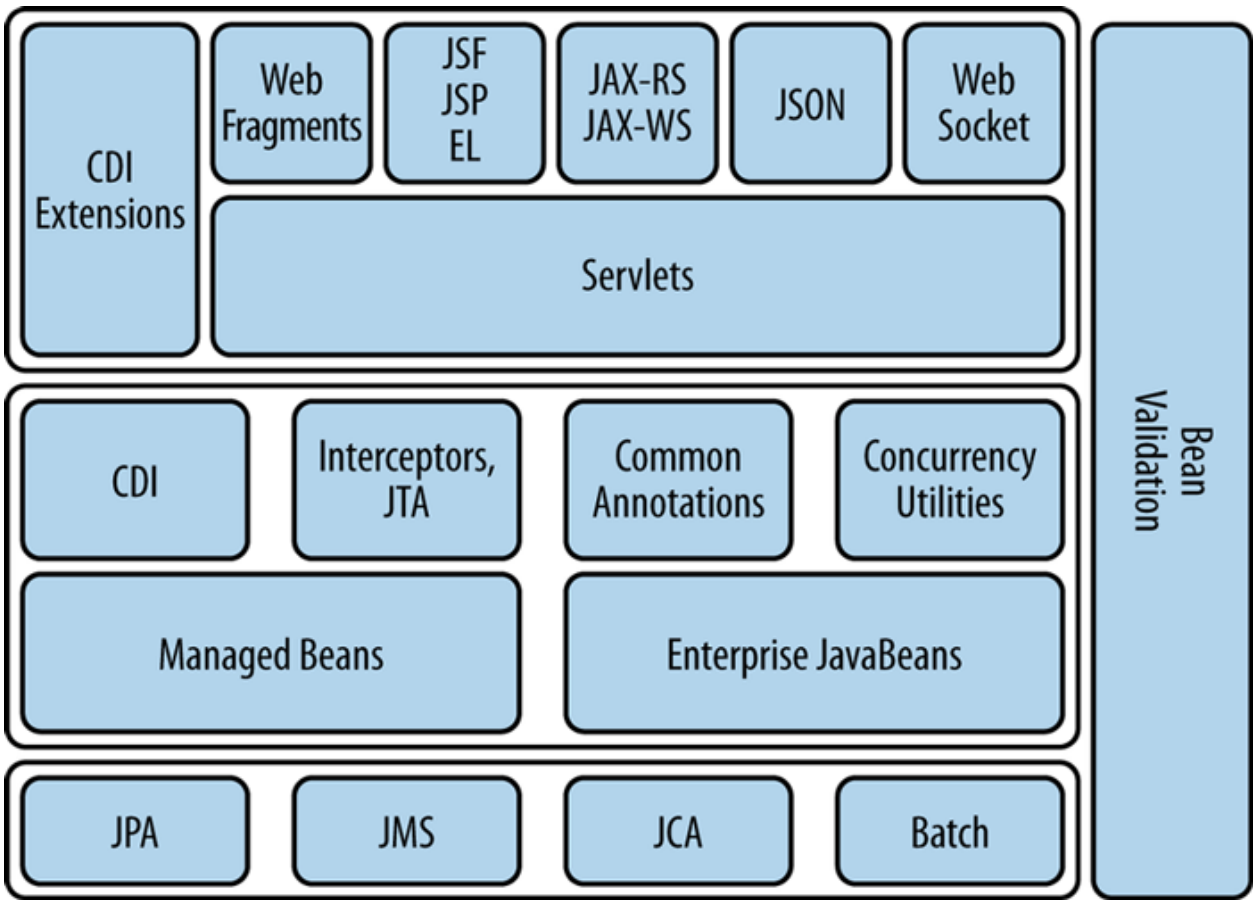


Рисунок 2.3. Розширена схема роботи Java EE додатку

2.2.2. Стек Spring

Розглянемо простий додаток на Spring стеку зображений на Рисунку 2.4. Як бачимо він не сильно-то й відрізняється від Java EE. Картинка відрізняється від попередньої за великим рахунком тільки більшою деталізацією. Так що середнє додаток на Spring стеці від програми Java EE не відрізняється практично ніяк. Єдина відмінність, яке відразу варто озвучити, - це те, що додаток на Java EE може працювати в загальному випадку тільки в рамках Enterprise Application Server'a (нагадую, що Tomcat ним не є), а додаток на Spring стеку може працювати на чому завгодно (на тому ж Tomcat) і навіть взагалі без сервера (так як запустить його всередині себе самостійно).

Це робить Spring стек ідеальним для реалізації елементів мікросервісної архітектури, але за рахунок відмови від підтримки всіх можливостей серверів додатків.

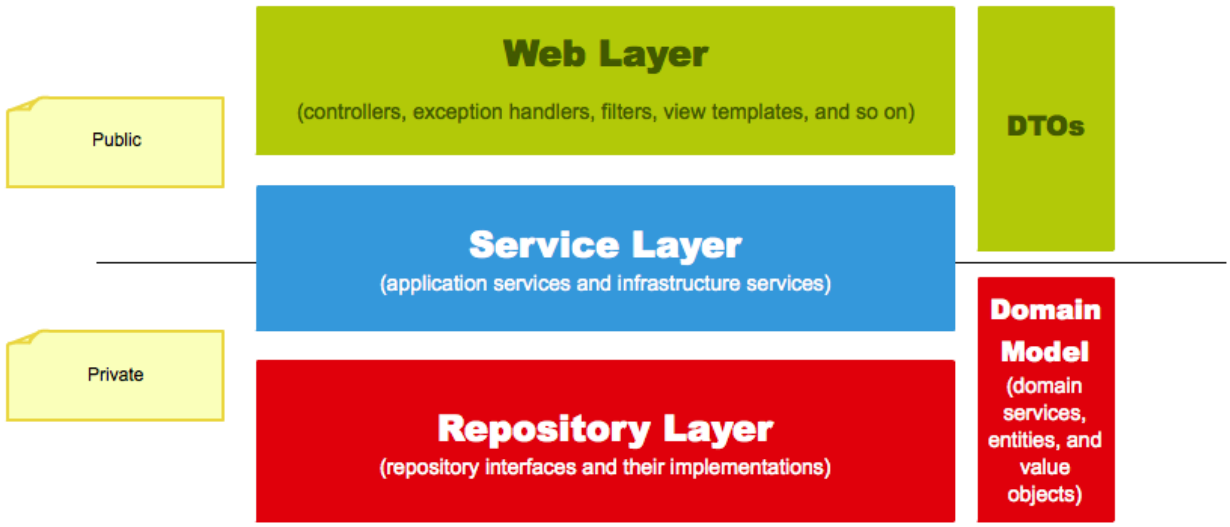


Рисунок 2.4. Спрощена схема Spring додатку

У загальному випадку кластерне додаток - це не про Spring, і питання масштабування для Spring додатки повинні вирішуватися окремо. У більшості випадків - істотно затратно.

Подивимося на весь стек бібліотек Spring зображений на Рисунку 2.5. Зараз бібліотек Spring надто багато, так що ми розглянемо основні. На сайті Spring до сих пір завантажена ось ця картинка, яка там знаходиться приблизно з 2014 року. Якщо придивитися, то все виглядає вельми схоже на те, що ми бачили в розділі Java EE. Насправді воно так і є. У всіх тих стеках, які існують на сьогоднішній момент є практично ідентичне уявлення про те, як треба писати Enterprise додатки. Кожен стек вважає, що об'єкти предметної області повинні бути Value Object, а сама логіка бізнес-шару повинна зберігатися окремо від них (в сервісах у Spring і в EJB у Java EE).

А це, як ви розумієте, - поділ даних і поведінку - ні разу не відповідає принципам ООП (але це вже інша тема для доповіді).

Це звичайний процедурний код з усіма своїми проблемами. Як мінімум ми не можемо використовувати жодного шаблону проектування.

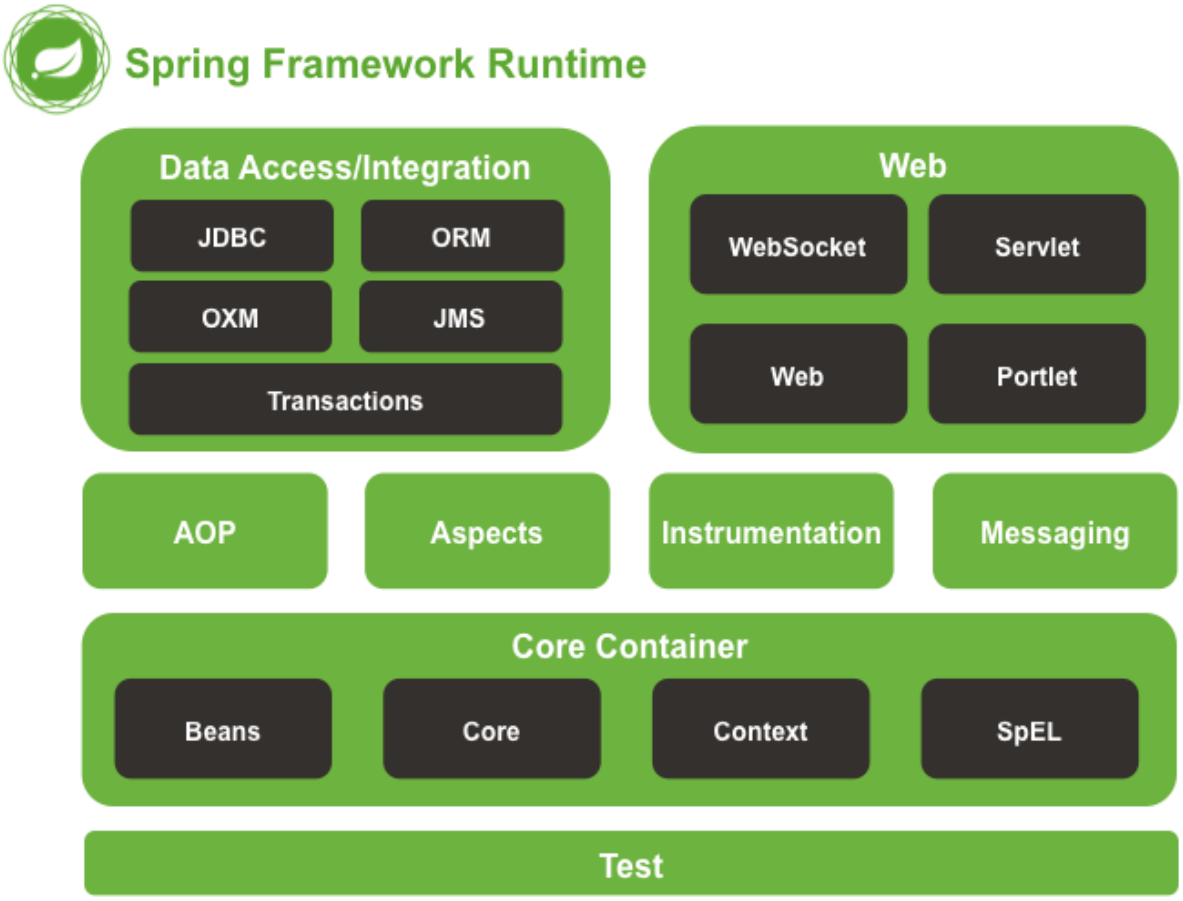


Рисунок 2.5. Основні модулі Spring фреймворку

2.2.3. Попарне порівняння стеків

До цього моменту ми порівняли тільки зовнішню архітектуру, зараз же перйдемо до більш глибоко порівняння. Розглянемо що ж хорошого і що поганого в кожному з них.

2.2.3.1. Spring DI (Dependency Injection) vs CDI (Context Dependency Injection)

Навіть з назви зрозуміло, що адра в цих стеках однакові. Так і є. Фактично в Java EE стеці спочатку і використовувалася спрінговий DI. Але після того як його включили прямо в Core, використовувати його окремо стало неможливо. Довелося робити свій DI з анотаціями.

Тому це й етап порівняння ми пропустимо, оскільки вони практично ідентичні.

2.2.3.2. Spring Beans vs EJB (Enterprise Java Beans)

На цьому пункті відмінностей істотно більше, простіше сказати що це геть дві протилежних речі. Spring beans - це просто звичайні джава біни, які можна кудись інжектити і нічого більше.

EJB - це досить потужний модуль, в яку вбудована підтримка розподіленого (мультинодного) виконання, включаючи розподілений збирач сміття, аутентифікацію, підтримку транзакцій і багато що ще. Все це виглядає доволі непогано, але от проблема якщо це все вам не потрібно, виходить що ви підключили дуже великий модуль, а використовуєте його на відсотків 8%. У зв'язку з цим виникають значні втрати в пам'яті та продуктивності.

2.2.3.3. Spring Service Locator vs JNDI

Тут теж ситуація аналогічна. Служба, яка допомагає знайти в тому ж JVM сервіс, і служба,яка може працювати на розподіленої системі і знаходити відповідність між чим завгодно. Включаючи прив'язку до LDAP компанії. Формально, призначені вони для одного і того ж, однак різниця дуже велика. Навіть на цьому етапі можна помітити закономірність: Spring це великий набір різних модулів де ви можете підключити конкретно те, що вам потрібно, а Java EE підключить вам відразу все.

2.2.3.4. @Async vs JMS

Асинхронність також реалізована в обох стеках на різному рівні. У стеці Java EE це окрема розподілена служба, що підтримує передачу повідомлень як точка-точка, а й багатьом одержувачам. Природно, відмовостійка, персистентная і так далі.

Як бачимо наше правило підтверджується. Щось просте і швидке проти розподіленої громадного та складного. Модель JMS зображено на Рисунку 2.7.

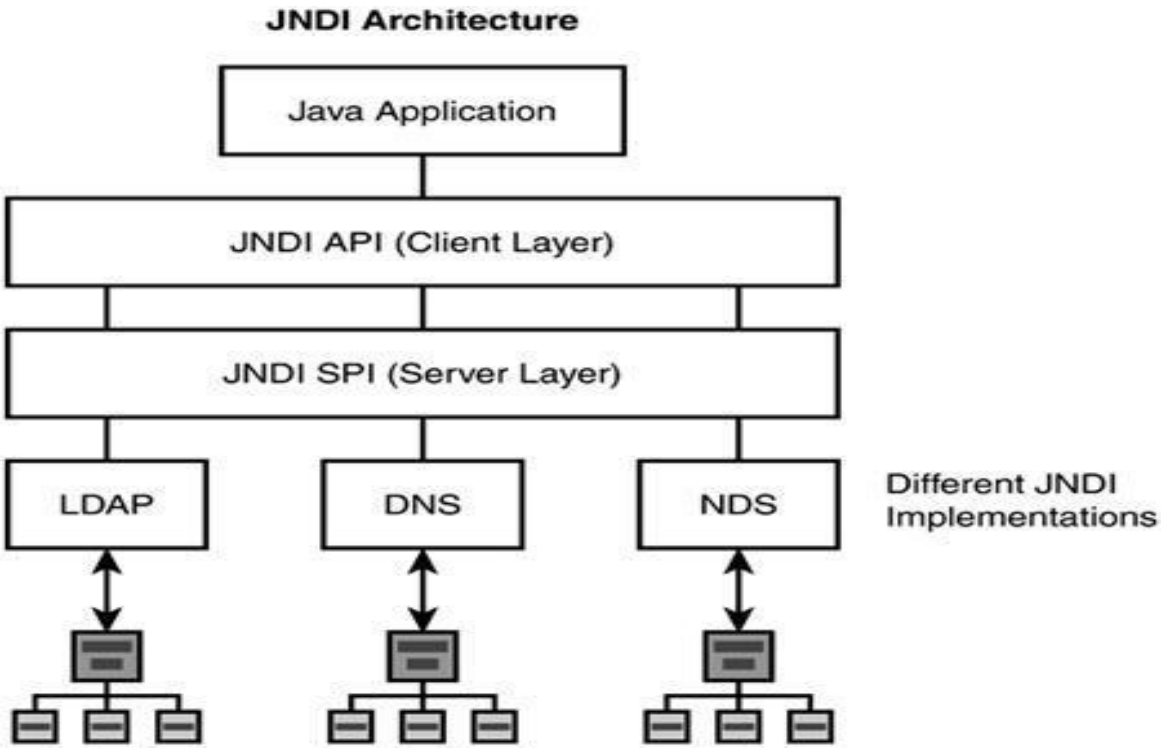


Рисунок 2.6. Архітектура JNDI

2.2.3.5. Spring MVC vs Java Server Faces (JSF)

Проблема цього порівняння в тому, що SpringMVC як темплейтного движка зазвичай використовується Thymeleaf або в більш сучасному варіанті - будь-якої Front-end движок (React, Angular & etc). У той час, як JSF - повноцінне GUI рішення з підтримкою AJAX з коробки і навіть SPA додатків за допомогою додаткових бібліотек, включаючи такі гіперзручні і красиві, як PrimeFaces і IceFaces. На цьому пункті тяжко сказати що краще, оскільки обравши Spring MVC ми маємо змогу вільно обирати Front-end частину, але якщо ми оберемо Thymeleaf(чого я не раджу), то на певному моменті ми обов'язково від нього відмовимось(за умови що проект буде розвиватися), оскільки у ньому є багато обмежень, проблем з підключенням AJAX. А обравши JSF ми знову ж таки отримаємо багато чого, не факт, що потрібного нам.

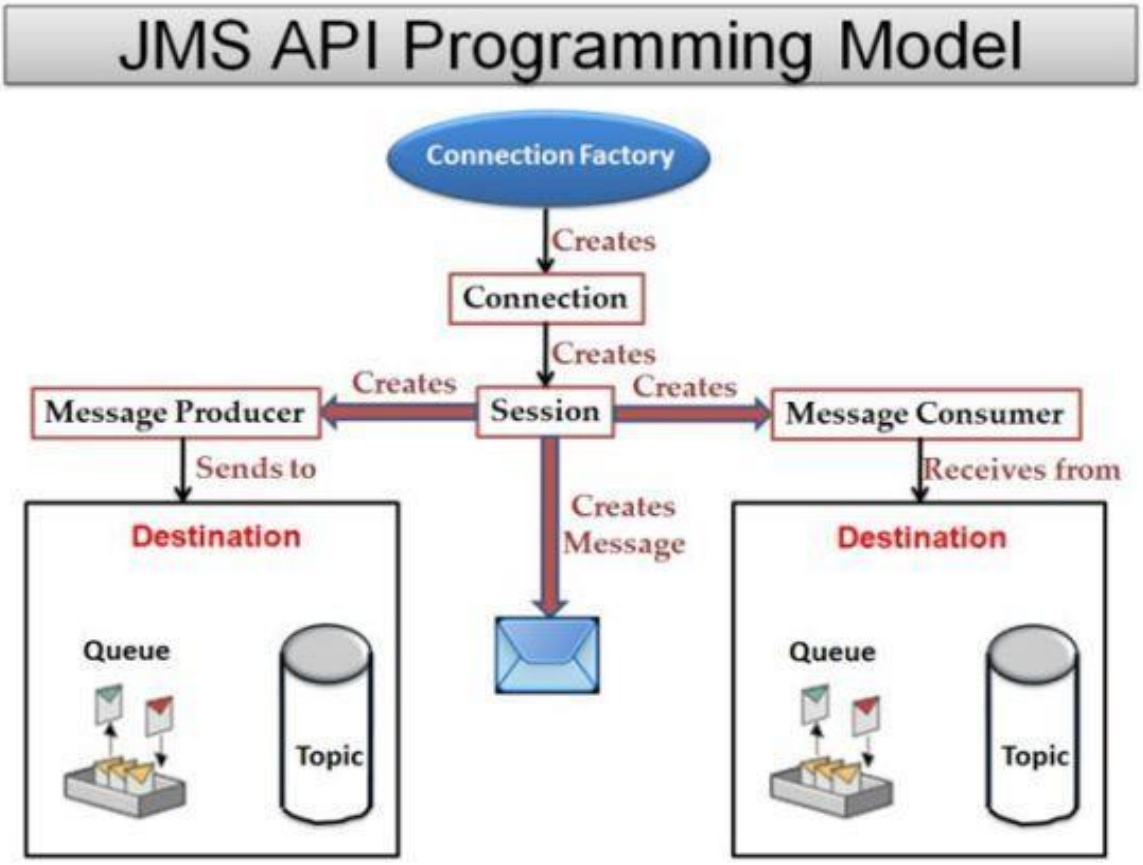


Рисунок 2.7. Модель JMS

2.2.3.6. Spring Data vs JPA

Це напевно єдиний пункт, в якому зі мною погодиться кожен досвідчений програміст. Spring просто непорівняно кращий за Java EE в цьому плані. Spring Data - це надзвичайно швидка і зручна річ, в якій ви можете створювати запити просто написавши назву метода . Аналогів в Java EE стеку.

2.2.3.7. Spring Security vs JAAS

Spring Security - фреймворк, який напевно є найскладнішим з усіх Spring фреймворків, з ним постійно виникають проблеми і на мою думку він має найгіршу документацію. У цьому пункті я віддаю перевагу JAAS.

JAAS, природно, підтримує розсподільність та багато іншого.

2.2.3.8. Spring Boot vs Enterprise Server

Spring Boot справді непогана річ, але чомусь більшість забувають , що він потрібен тільки для того, що без нього дуже складно зібрати докупи всі ті Spring модулі. Java EE такого не потребує, аналогом для Spring Boot є просто будь-який Enterprise server. Там вже є всі потрібні фреймворки потрібних і сумісних версій, вже налаштовані для роботи один з одним.

2.3. Тестовий модуль

При розробці продуктів часто недооцінюють важливість тестів, але саме тести разом з стеком технологій, який підтримує модульність, допомагає уникнути безліч проблем при розширенні чи зміні функціоналу. У нашому, для написання тестового модуля, будуть використані бібліотеки Junit та фреймворк Mockito.

Рисунок 2.8. Зображення емблеми Junit5

Junit - це бібліотека для модульного тестування на Java. Ми будемо використовувати два пакети цієї бібліотеки: junit.framework.Assert та junit.framework.TestCase. Assert включає в себе функіонал для порівняння величин та об'єктів: assertEquals, assertNotNull, assertSame… TestCase володіє функціоналом для run, setUp, tearDown.

Mockito - це фреймворк, який дозволяє перекрити реалізацію одного методом реалізацією іншого. Наприклад при тестуванні реєстрації ми не маємо змогу переходити на пошту щоб підтвердити її, зачасту такого електронного адресу навіть не існує. Для взагалі здійснити реєстрацію в тестовому режимі нам допоможе Mockito. За допомогою нього ми перекриємо метод активації пошти.

Рисунок 2.9. Зображення емблеми Mockito

2.4. Розробка алгоритму

Будь-який додаток для пошуку вимагає для виконання певні операцій. Було розроблено алгоритм пошуку для нашої системи. Список операцій пошуку наведено в Таблиці 1.1. Під час роботи додатку , база даних буде наповнюватися відповідними даними. Ці дані будуть змінені в формат документів які в свою чергу будуть проаналізовані, та індексовані. Для побудови запитів буде написано зручне Api, де ми будемо керуватися тільки вхідними даними, запити будуть будуватися автоматично .

Таблиця 1.1. Алгоритм роботи пошукової системи

Крок	**Заголовок**	**Опис**
1	Отримати дані	Найпершим кроком будь-якого пошукового додатка є збір цільового вмісту, за яким буде проводитися пошук.
2	Побудувати документ	Наступним кроком є створення документа (ів) з необробленого вмісту, який пошукове додаток може легко зрозуміти і інтерпретувати.
3	Проаналізувати документ	Перш ніж почати індексацію, документ повинен бути проаналізований.

Продовження таблиц1 1.1. Алгоритм роботи пошукової системи

Крок	**Заголовок**	**Опис**
4	Індексація документа	Після того, як документи побудовані і проаналізовані, наступним кроком є їх індексація, щоб цей документ можна було отримати на основі певних ключів, а не всього вмісту документа. Індексування аналогічно індексам, які ми маємо в кінці книги, де звичайні слова показані з номерами їх сторінок, так що ці слова можна швидко відстежувати, а не шукати в повній книзі.
5	Користувальницький інтерфейс	Коли база даних індексів готова, додаток може виконувати пошукові операції. Щоб допомогти користувачеві виконати пошук, додаток повинен надати призначений для користувача інтерфейс, в якому користувач може вводити текст і ініціювати процес пошуку.
6	Побудувати запит запрос	Як тільки користувач відправляє запит на пошук тексту, додаток повинен підготувати об'єкт запиту, використовуючи цей текст, який потім можна використовувати для запиту бази даних індексу для отримання відповідних відомостей.
7	Пошуковий запит	Використовуючи об'єкт запиту, перевіряється база даних індексу для отримання відповідних відомостей і документів вмісту.
8	Результати рендерингу	Як тільки необхідний результат отриманий, додаток повинен вирішити, як відобразити результати користувачеві, використовуючи його для користувача інтерфейс.

На рисунку 2.10 показано загальне зображення того, як працює наша пошукова система. Зображена взаємодія між користувачем, побудова запиту, обробка запиту пошук даних по індексу , пошук відповідного документу, та відображення результату пошуку.

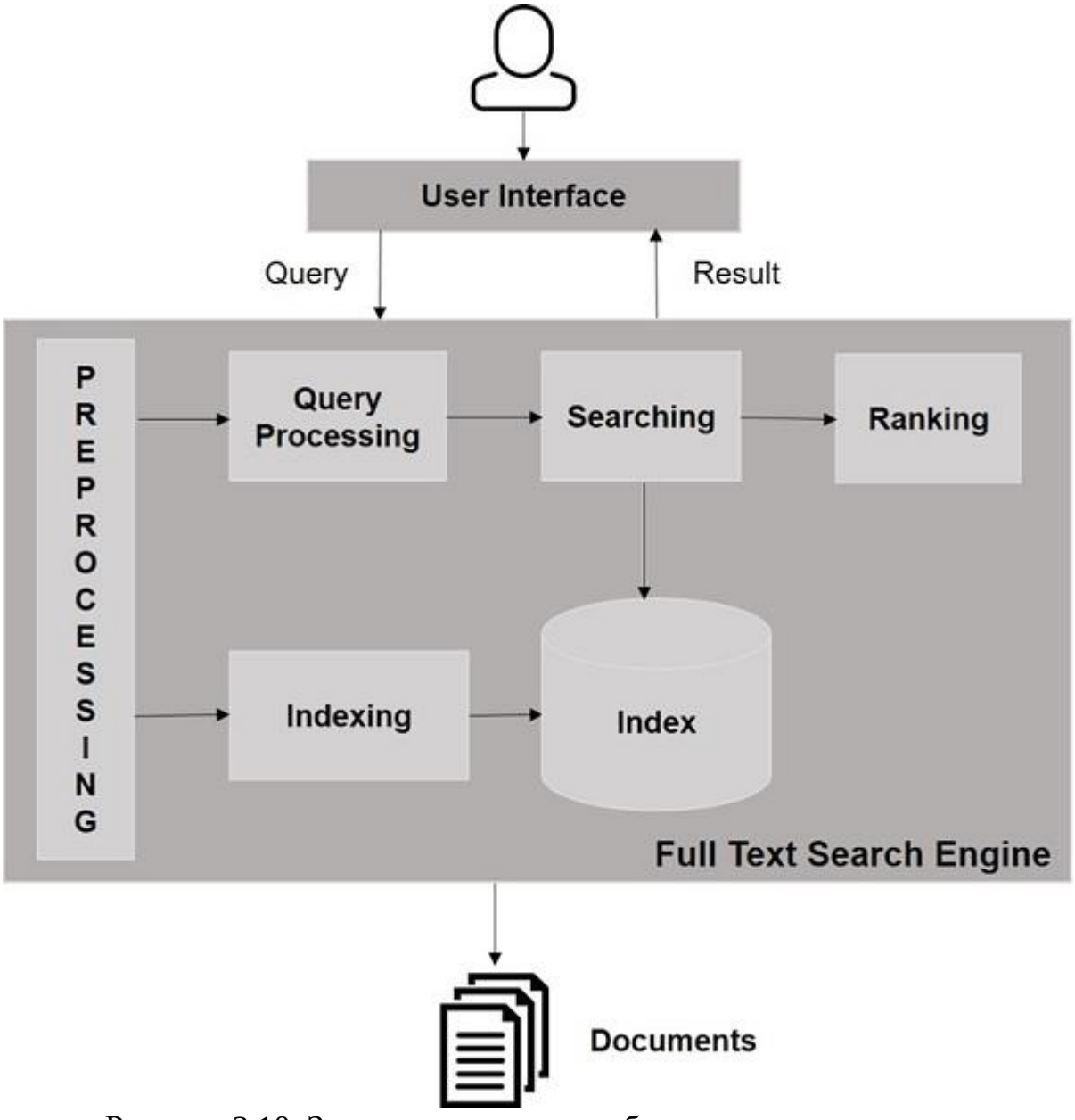


Рисунок 2.10. Загальне структура роботи системи пошуку

ВИСНОВКИ ДО РОЗДІЛУ 2

У даному розділі було детально розглянуто кожне з можливих рішень наших проблем. Проведено аналіз та порівняно кожну технологію. На основі отриманих даних було складено такі висновки:

- Java EE - стек на якому зараз існує більшість серверів, що створює більший попит на саме цей стек. Внаслідок цього стек постійно оновлюється, створюються нові технології та додатки і існує безліч спеціалістів хорошого рівня володіння стеком Java EE. Та змушує нас використовувати безліч непотрібни речей, цим знижуючи швидкодію та продуктивність продуктів.
- Spring - це більш сучасне рішення, має безліч потрібних фреймворків, та не обтяжує користувача непотрібним функціоналом. Але є більш фінансово затратним ніж Java EE.
- Реляційні бази даних - у зв'язку з більшою популярністю має безліч документації, інструментів для розробки. Практично відсутній пошук за текстом.
- Solr - практично відсутня документація, але висока якість реалізації та чудовий текстовий пошук.

Враховуючи особливості поставленої задачі та порівняння стеків було спроектовано онлайн бібліотеку реалізацією якої є веб додатку з комбінація Spring та Solr. Було визначено необхідні фреймворки Spring та способи підключення API та БД. Визначено необхідний функціонал та спроектовано каркас тестів.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Огляд призначення розроблюваної системи

В даній роботі буде розроблена система онлайн бібліотеки. Дана система повинна надавати користувацький інтерфейс для читання, завантаження книг.

Користувачам буде надана можливість реєстрації та подальшої авторизації. Авторизовані користувачі можуть оцінювати книги та залишати коментарі. Також на сайті буде широка система фільтрів книг, та можливість зберігати книги до обраних.

Як вимоги до основного функціоналу клієнта можна виділити наступні пункти:

1) перегляд списку книг;
2) фільтрування книг
3) перегляд детальної інформації про книг;
4) можливість оцінити книгу та залишити до неї відгук;
5) переглянути список інших книг цього автора;
6) пошук книг за ключовими словами, авторами;
7) сортування книг за певними критеріями;
8) форма реєстрації користувачів;
9) форма логіну;

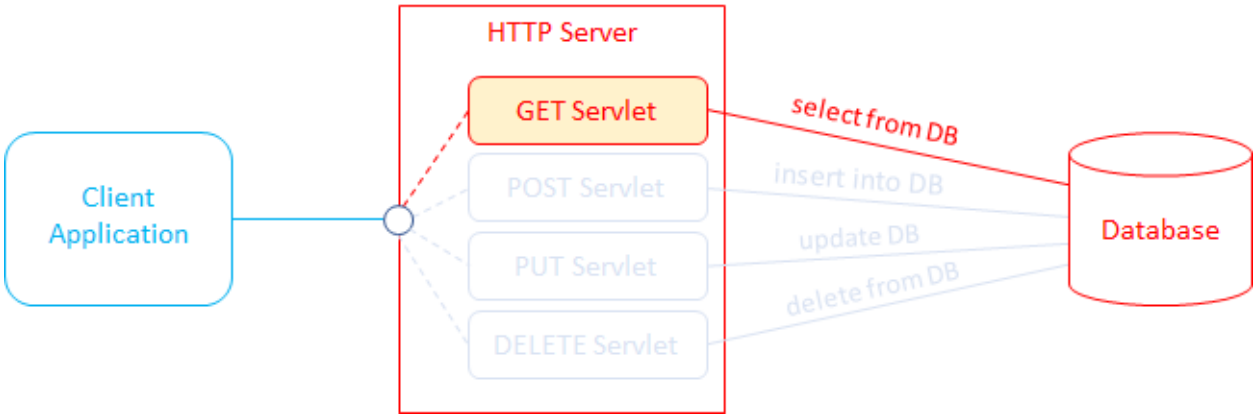


Рисунок 3.1. Взаємодія елементів верхнього рівня системи

3.2 Огляд технологічної бази

Наша система онлайн бібліотеки є доволі складним програмним рішенням, тому вимагає сучасних технологій, які надають відповідний функціонал. Розглянемо технології які будуть використані в процесі розробки.

3.2.1 Java

Java - як основна мова розробки. Java — об'єктно-орієнтована мова програмування, створена компанією Sun в 1995.

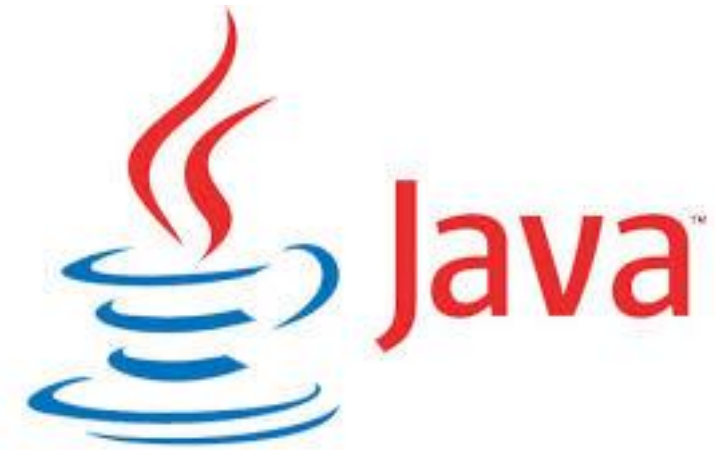


Рисунок 3.2. Логотип Java

Компанія «Oracle» у 2009 викупила компанію Sun, та з того моменту вона займається розробкою Java. В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Обираємо цю мову через її простоту надійність та популярність. У зв'язку з її популярністю безліч користувачів використовують джава, безліч великих компаній будуть свої сервера на Java, що призводить до збільшення попиту на сучасне якісне API для Java.

3.2.2 Spring

Spring - якнайкраще рішення проблеми з модульністю сучасних сайтів. У цьому фреймворці ми знайдемо все, що нам потрібне для нашого проекту. А саме :

Spring MVC - За допомогою нього ми реалізуємо архітектуру модель-уявлення-контролер (model-view-controller). Spring забезпечує готові

компоненти, які можуть бути використані (і використовуються в нашому проекті) для розробки веб-додатків.

Головною метою MVC є поділ об'єктів, бізнес-логіки й зовнішнього вигляду програми. Всі ці компоненти слабо пов'язані між собою і при бажанні ми можемо змінити, наприклад, зовнішній вигляд програми, не вносячи суттєві зміни в інші два компонента. Це

Spring Security - За допомогою цього фреймворку, ми реалізуємо в нашому проекті аутентифікацію, та авторизацію в онлайн бібліотеці. Як і всі Spring проекти, справжня сила Spring Security в тому, що він може бути легко доповнений за потреби функціоналом.

Комплексна і розширюється підтримка як аутентифікації, так і авторизації. Захист від атак типу фіксація сесії, клікджекінг, міжсайтовий підробка запиту і ін.

Spring Data - (а саме Spring Data Solr) за допомогою цього фреймворку ми з легкістю пов’яжемо нашу Онлайн Бібліотеку з Apache Solr. Фреймворк володіє додатковим механізм для взаємодії з сутностями бази даних, організації їх в репозиторії, вилучення даних, зміна.

Spring Boot - за допомогою цього фремворку ми з легкістю об'єднаємо всі попередньо вказані фреймворки в одне ціле. У Spring Boot було включено Spring-платформу і сторонні бібліотеки, щоб ми могли запустити з мінімум зусиллями. Більшості Spring Boot додатків потрібно зовсім маленька Spring-конфігурація, але це не наш випадок , оскільки нам потрібно підключити Solr. Також Spring Boot володіє вбудованими Tomcat та Jetty.

3.2.3 Solr

Як найкраще вирішення проблеми збереження та пошуку по великим об'ємах даних. Для реалізації пошуку за рядками, пов’язаними словами жанрами, авторами.

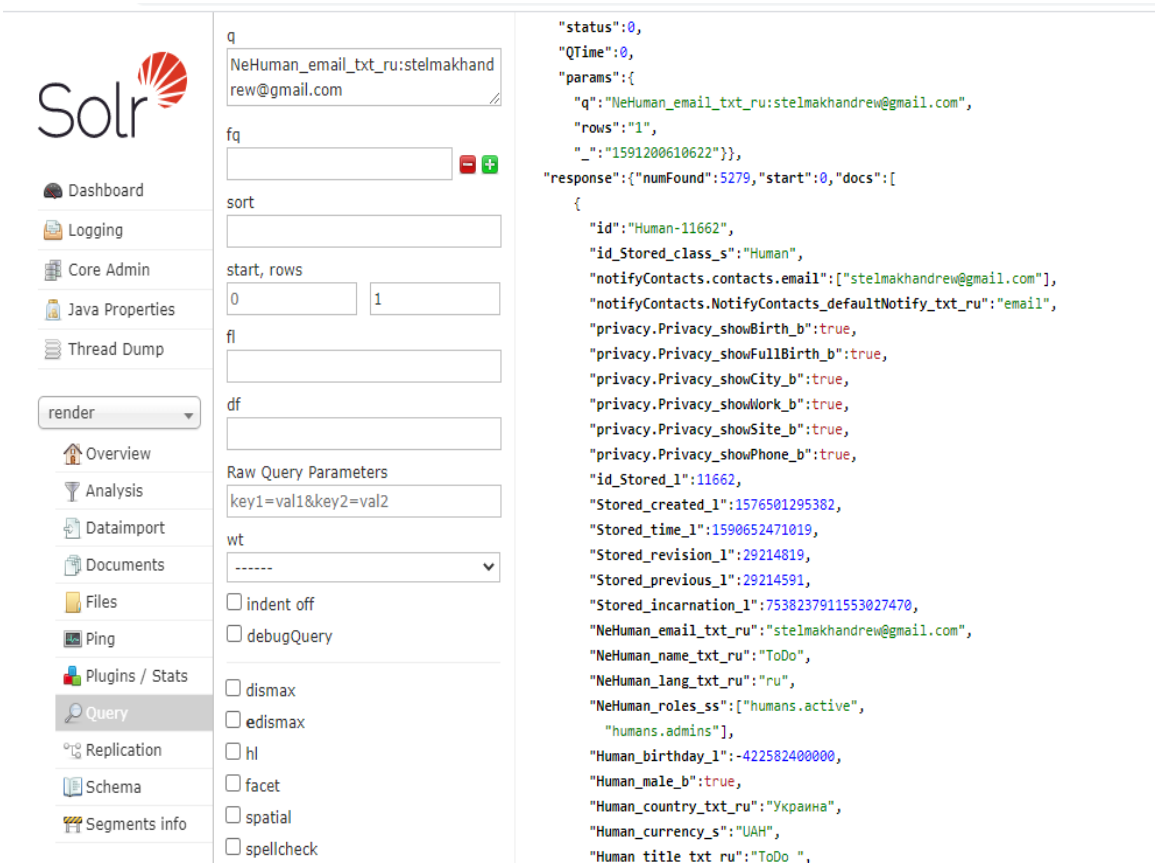


Рисунок 3.3. Приклад інтерфейсу адміністратора Solr

3.2.4 JS CSS HTML

Розроблювана система буде мати користувацький інтерфейс. Для його написання оберети фреймворків не будемо, використаємо звичайний набір інструментів такий як JS CSS HTML. Оскільки наша проблема ніяк не пов'язані з користувацький інтерфейсом.

3.2.5 Thymeleaf

Thymeleaf - сучасний серверний механізм Java-шаблонів для веб-і автономних середовищ, здатний обробляти HTML, XML, JavaScript, CSS і навіть простий текст.

Основною метою Thymeleaf є створення елегантного і зручного способу шаблонізації. Щоб досягти цього, Thymeleaf грунтується на концепції Natural Templates, щоб впровадити свою логіку в файли шаблонів таким чином, щоб цей шаблон не впливав на відображення прототипу дизайну. Це покращує комунікацію в команді і зменшує розрив між дизайнерсько-програмістські

групами. На Рисунок 3.4. можна побачити, як за допомогою Thymeleaf можна вписати Java вираз прямо в html. Так на рисунку видно, як за допомогою Java змінюється формат подачі рейтингу книги, в залежності від того чи авторизований користувач, чи ні.

```
<img th:src="${book.getImage()}" style="...">
<div th:text="${book.getAuthor()}"/>

<div th:if="${!user}">
    <div style="..." th:text="@{${book.getRating()}+'/10.0 (' + ${book.getCountRating()}+ ')'}"></div>
</div>

<form th:if="${!user}" th:action="@{'/books/' + ${book.getId()}}" th:method="PATCH">
    <div class="rating">
```

Рисунок 3.4. Можливості Thymeleaf

Thymeleaf також був розроблений з самого початку з урахуванням стандартів Web, особливо HTML5, що дозволяє вам створювати повністю відповідають стандарту шаблони.

3.3. Бізнес-кейси

3.3.1. Failed Login

При введені користувачем некоректних даних, в апі відбувається валідація даних та запити в базу даних, для пошуку користувача з відповідною електронною поштою(яка є унікальною). Якщо користувача не знайдено в відповіді разом зі статусом 401, вказується причина , а саме “Помилкова

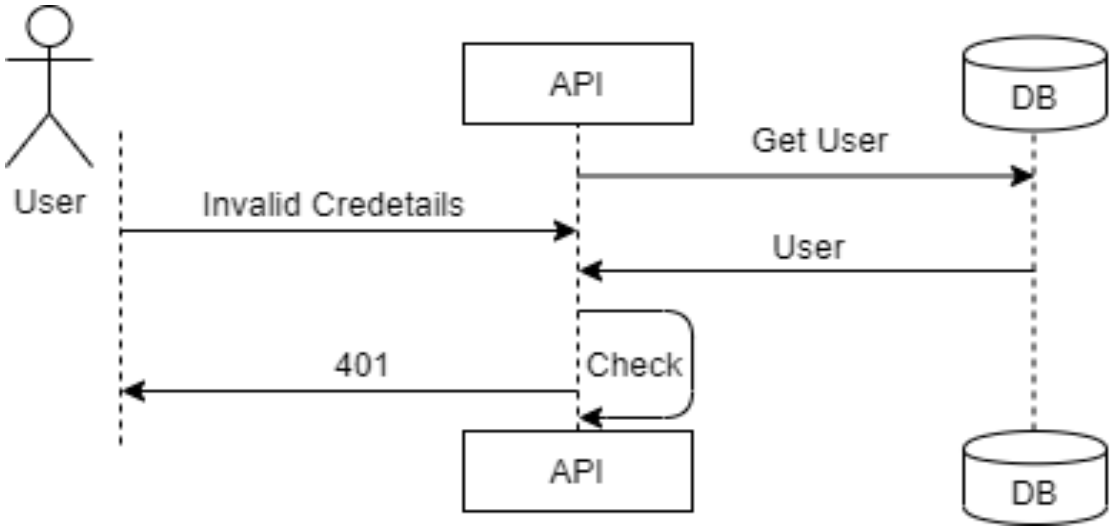


Рисунок 3.5. Алгоритм дій при помилковій авторизації

адреса”, якщо ж користувач існує , але пароль вказаний невірно , то повертається помилка “Пароль вказано невірно”.

3.3.2. Успішний Login

При введені користувачем коректних даних, які пройшли валідацію, то в відповіді вказується статус 200 та відбувається логін користувача.

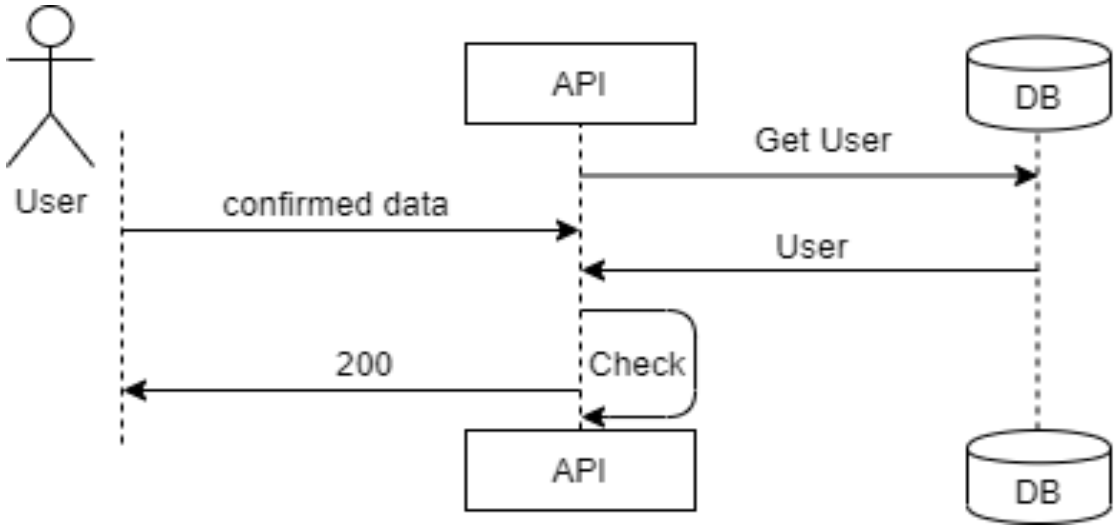


Рисунок 3.6. Алгоритм дій при успішній авторизації

3.3.3. Відкриття сторінки книги

При такій, здавалося б простій дії, як відкриття сторінки з книгою, насправді відбувається складних механізм. А саме при запиті Get /book/{id} , дістається сесія користувача, і перевіряється чи користувач залогінений, якщо ні - то користувач не має змоги виставити рейтинг та залишити відгук. Якщо ж користувач залогінений - відбувається ще один запит в базу даних для перевірки чи виставляв вже рейтинг даний користувач. В будь якому з цих випадків в відповіді вказується статус 200(оскільки обидва випадки є прийнятними та вважаються успішними), але також відповіді відправляється флаг , який вказує фронтенду чи потрібно давати змогу залишити коментар чи ні. Разом з запитом на API надходить сесія користувача, з якої ми дістаємо об’єкт класа Human. Наступним кроком робиться запит в базу даних, далі в залежності від того чи існує користувач чи ні, перевіряється його можливість залишати відгуки. Незалежно від цих кроків статус відповіді буде 200. Оскільки кожні з цих

ситуацій є валідною та припустимою. Http статус 200 вказує на те, що ресурс на який було подано запит успішно знайдено та передано в тілі відповіді.

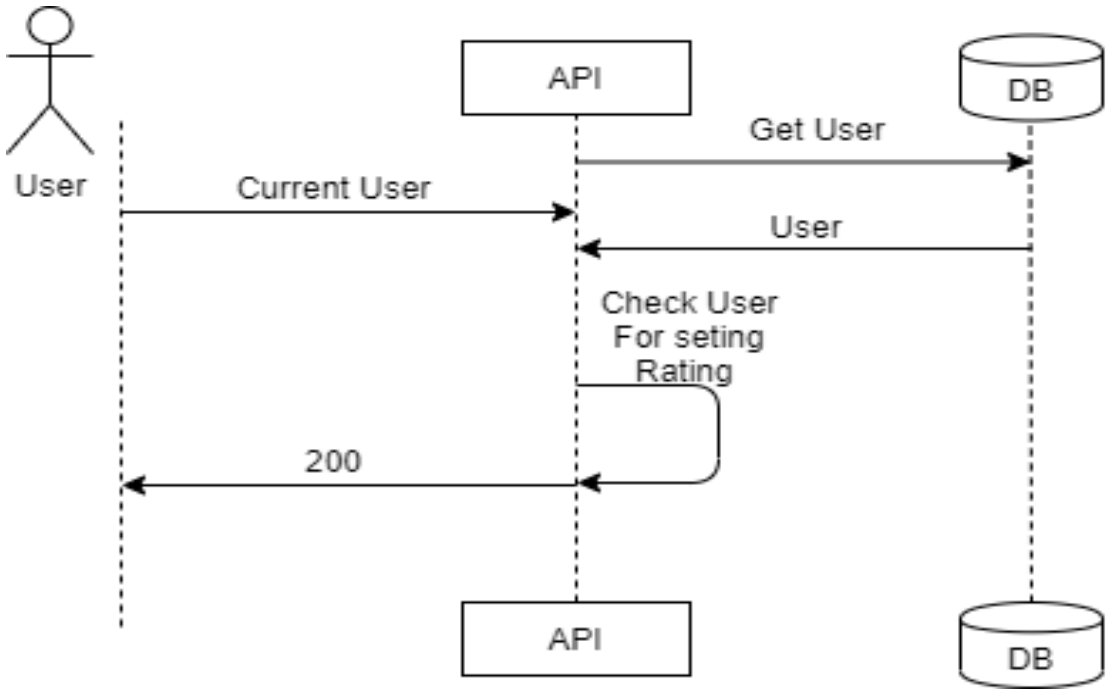


Рисунок 3.7. Алгоритм дій при відкритті сторінки з книгою

Нижче наведена реалізація відображення цього бізнес кейсу, та відмінності в залежності від отриманих відповідей. Так на Рисунок 3.8. зображена ситуація для неавторизованого користувача. Неавторизований користувач немає змоги виставляти рейтинг, або залишати коментарі. Коментарі зборонено залишати, щоб не було можливості безкарно спамити, або флудити. А рейтинг для того, щоб не мати змоги штучно понижувати, або підвищувати рейтинг книги. Також неавторизований користувач немає змоги додати книгу до улюблених книг. Улюблені книги відображаються на сторінці користувача і він має посилання на них, для швидкого доступу. Для неавторизованого користувача виводиться відображення тільки рейтингу книги. А на Рисунку 3.9.зображено інтерфейс для авторизованого користувача, вже з можливістю відгуків та виставлення рейтингу. Можливість виставляти рейтинг для книги один користувач має змогу тільки один раз. Це контролюється одним додатковим запитом в базу, який перевіряє чи виставляв користувач рейтинг для цієї книги, чи ні.

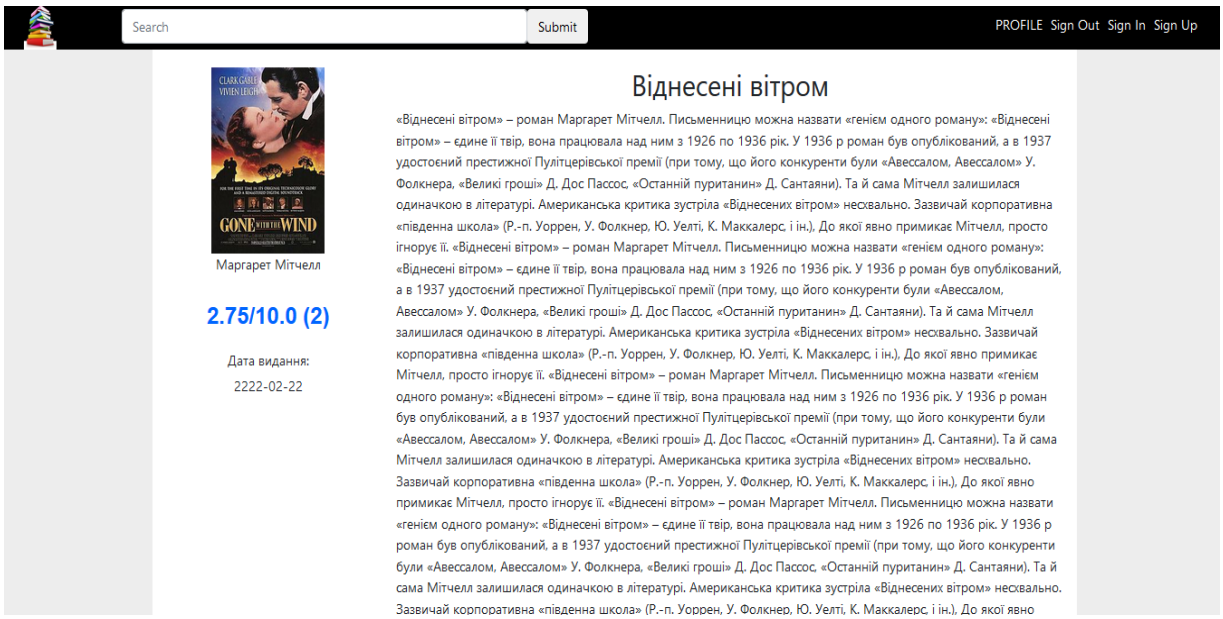


Рисунок 3.8. Особливості інтерфейсу неавторизованого користувача

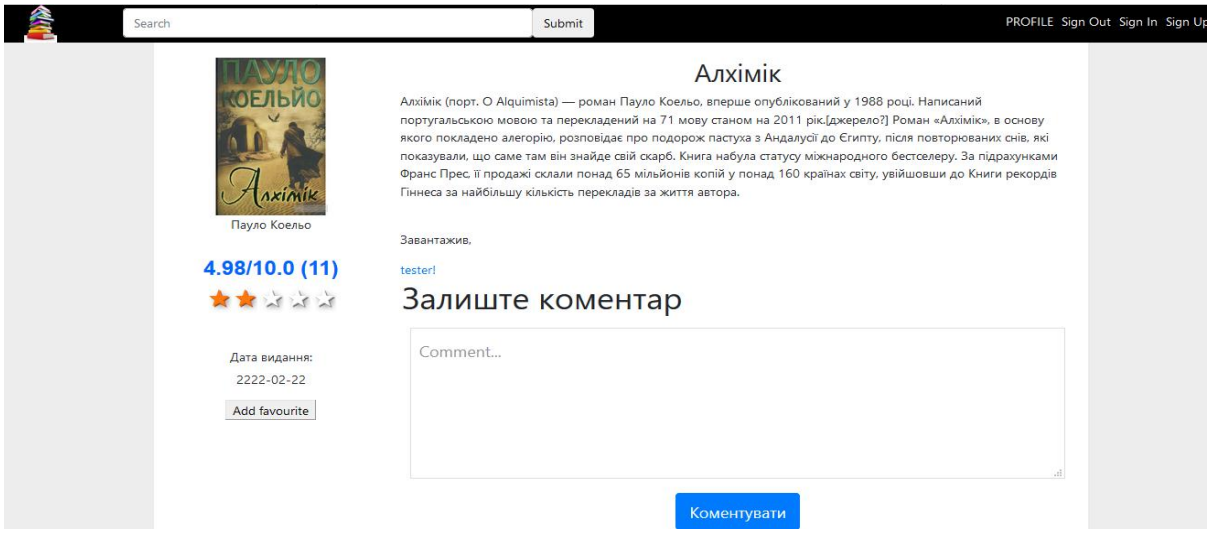


Рисунок 3.9. Особливості інтерфейсу авторизованого користувача

3.3.4 Реєстрація

При реєстрація відправляються дані на апі, де вони валідуються та де відбувається запит до бази даних і перевірка на унікальність даних. При успішному проходженні перевірки повертається статус код 201 та відбувається перенаправлення на сторінку створеного користувача.

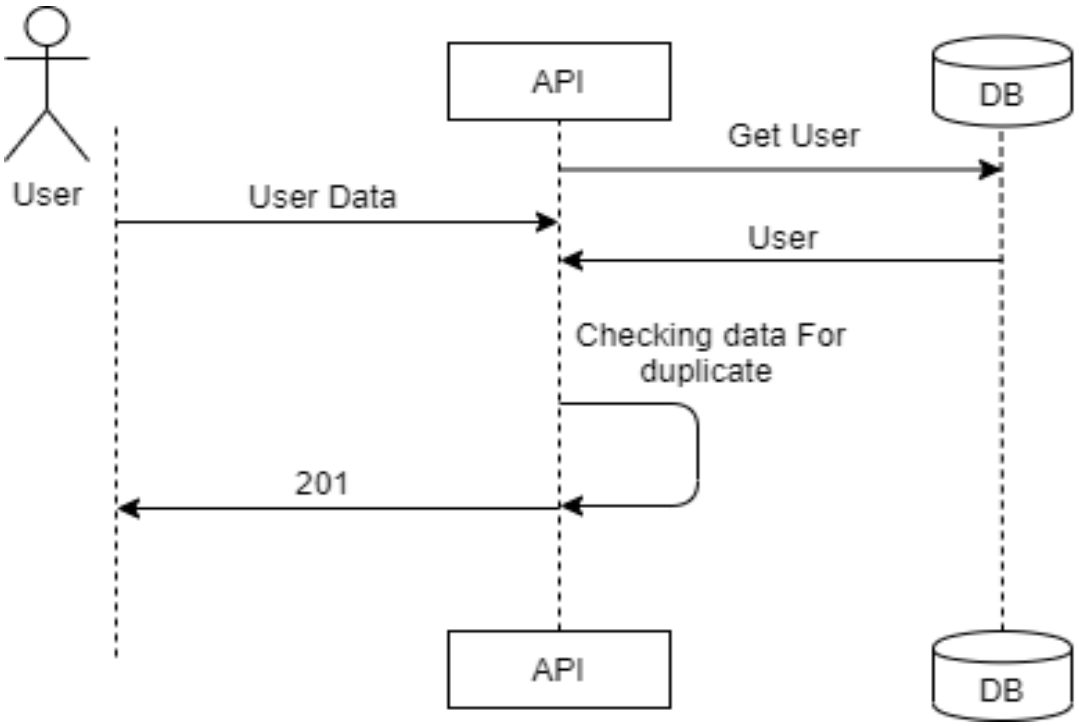


Рисунок 3.10. Алгоритм реєстрації

На Рисунок 3.11. можна побачити примітивну реалізацію відображення реєстрації нашої онлайн бібліотеки. Реєстрація потребує унікального електронного адресу, та Username .

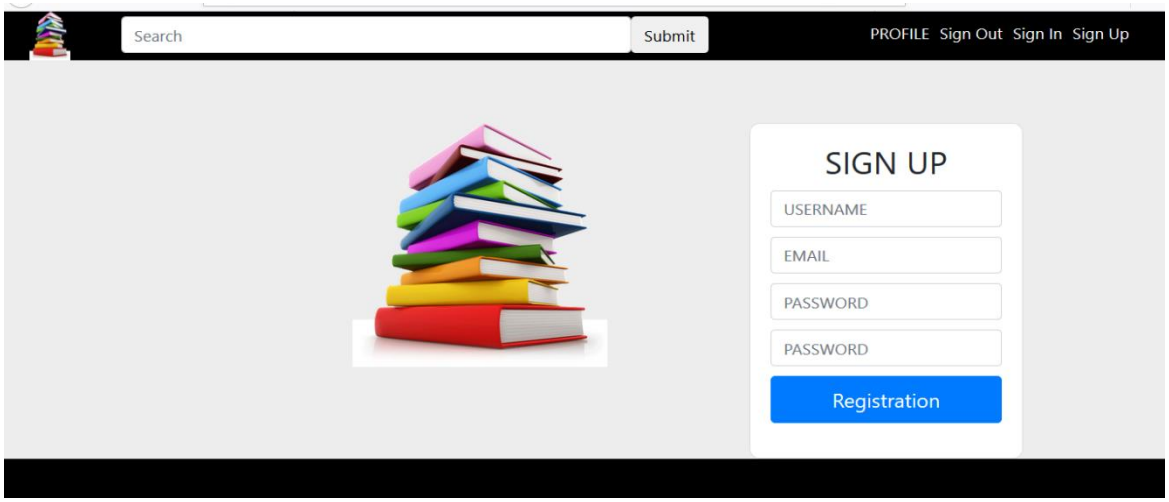


Рисунок 3.11. Форма реєстрації

3.4. Тестування

3.4.1. Терміни

Тестування - це невід'ємна частина розробки продуктів. Існує два підходи розробки програмного продукту:

- Написання спочатку коду додатку, а потім спроби покрити код тестами.
- Test-driven development (TDD) - у такому випадку розробка розбивається на дрібні модулі. Перед розробкою певного модуля створюється тест для перевірки нормальної роботи цього модуля, а тільки потім пишеться сам модуль.

Другий спосіб вважається більш професійним. На жаль, через брак часу, не було можливості розбивати весь додаток на дрібні модулі, тому у нашому випадку писався спочатку код, а потім частина функціоналу покривалась тестами.

3.4.2. Типи тестування

Існують десятки видів тестування, розглянемо види тестування програмного забезпечення які були проведені :

- Функціональне тестування - тестування певних функцій(авторизація користувача, можливість залишати відгуки). Функціональне тестування буває двох видів: “white box” та “black box”. White Box - це тестування за якого відомо внутрішню систему реалізації певної функції. Black Box - це тестування за якого реалізація певної функції являється прихованою і ми керуємося тільки вхідними та вихідними даними.
- Системне тестування - це тестування всієї системи відразу.
- Тестування продуктивності - це тестування за якого визначають швидкість роботи системи за різних умов. Тестування під навантаженням - це тестування за стандартних умов роботи додатку. Стресове тестування - це тестування за якого визначаються найгірші умови, за яких система може працювати.

- Модульне тестування - це тестування за якого перевіряється конкретний модуль програми.

3.5. Приклади тестування системи

Для тестування основних функцій, таких як оновлення, видалення та створення об'єктів, було створено абстрактний клас для тестування AbstractTest.java. У цьому класі було проведені всі початкові конфігурації для створення пустої тестової бази та тестового користувача з правами адміністратора, які зображені на Рисунок 3.12.

```
@BeforeEach
public void ost() {
    mockMvc = webAppContextSetup(context)
//                .apply(springSecurity())
            .build();
    setResource();
    controllerMapping = "/" + resource.getClass().getAnnotation(RequestMapping.class).value()[0];

    human = humanService.getById(1);
    if (human == null || !humanService.check(RoleService.ADMIN, human)) {
        human = humanService.createHuman( email: generateRandomString( length: 6) + "@springbook.net",  name: "Test",  surname: "
        humanService.enable(RoleService.ADMIN, human);
        humanService.enable(RoleService.ACTIVE, human);
        humanService.activate(human);
    }
}
```

Рисунок 3.12. Метод для підготовки проекту до тестування

3.5.1. Create

Для тестування функції збереження було створено універсальний функціональний тест для будь якого об'єкта. Параметр controllerMapping вказує на потрібний url для запиту для конкретного об'єкта. Відправивши запит Http PUT по заданному url отримується відповідь ,перевіряється коректність отриманих даних та перевіряється наявність створеного об'єкта в базі даних. Метод зображено на Рисунок 3.13.

3.5.2. Update

Для тестування функції оновлення було створено універсальний функціональний тест для будь якого об'єкта. Відправивши запит Http PUT по заданному url отримується відповідь ,перевіряється коректність отриманих даних та перевіряються дані, які зберігаються в базі на наявність оновлення об'єкта. Метод зображено на Рисунок 3.14.

```
@Test
public void checkForCorrectCreateSpawn() throws Exception {
    if (!(resource instanceof GenericResource)) return;
    GenericService service = resource.getService();
    String body = getSpawnJson().toString();
    String response = mockMvc.perform(put( urlTemplate: controllerMapping + "/spawn")
            .contentType(MediaType.APPLICATION_JSON)
            .content(body)
            .session(createTestSessionWithAdmin(human)))
            .andExpect(status().is( status: 201)).andReturn().getResponse().getContentAsString();
    JsonNode id = mapper.readTree(response).get(ID);
    assertTrue(id.canConvertToLong());

    if (resource instanceof CursorSupport || resource instanceof FilterSupport)
        assertTrue(checkForContainsId(id.asLong()));
    assertNotNull(service.getById(id.asLong()));
}
```

Рисунок 3.13. Метод перевірки збереження об'єкта

3.5.3. Delete

Для тестування функції видалення було створено універсальний функціональний тест для будь якого об'єкта. Відправивши запит Http DELETE по заданному url отримується відповідь, перевіряються дані, які зберігаються в базі на відсутність об'єкта. Метод зображено на Рисунок 3.15.

```
@Test
public void checkForCorrectUpdateSpawn() throws Exception {
    if (!(resource instanceof GenericResource)) return;
    setUpdateDate();
    GenericService service = resource.getService();
    Entity t = service.create((s) -> {});
    JsonNode node = getSpawnJson();

    node = ((ObjectNode) node).put(ID, t.getId());
    node = ((ObjectNode) node).put(TEST_FIELD, NEW_VALUE_TEST_FIELD);

    String response = mockMvc.perform(put( urlTemplate: controllerMapping + "/" + t.getId())
            .contentType(MediaType.APPLICATION_JSON)
            .content(node.toString())
            .session(createTestSessionWithAdmin(human)))
            .andExpect(status().is( status: 200)).andReturn().getResponse().getContentAsString();
    JsonNode nodeResult = mapper.readTree(response);
    assertEquals(NEW_VALUE_TEST_FIELD, nodeResult.get(TEST_FIELD).textValue());
}
```

Рисунок 3.14. Метод перевірки оновлення об'єкта

```java
@Test
public void checkForCorrectDeleteSpawn() throws Exception {
    if (!(resource instanceof GenericResource)) return;


    GenericService service = resource.getService();
    Entity t = service.create((s) -> {});

    if (resource instanceof CursorSupport || resource instanceof FilterSupport)
        assertTrue(checkForContainsId(t.getId()));
    assertTrue(service.all().contains(t));

    mockMvc.perform(delete( urlTemplate: controllerMapping + "/" + t.getId())
            .session(createTestSessionWithAdmin(human)))
            .andExpect(status().is( status: 200)).andReturn().getResponse().getContentAsString();

    if (resource instanceof CursorSupport || resource instanceof FilterSupport)
        assertFalse(checkForContainsId(t.getId()));
    assertFalse(service.all().contains(t));
}
```

Рисунок 3.15. Метод перевірки видалення об'єкта

3.5.4. Клас для тестування дії з базою даних

Розглянемо реалізацію метода тестування основних функцій на прикладі обєкта Human. Як видно на Рисунок 3.16. реалізація основних методів маніпуляцій з базою даних займає всього 10 рядків, чого було досягнуто за рахунок створення абстрактного метода для тестування ресурсів. Усе що потрібно це перевизначити метод, який вказує потрібний ресурс, та вказати дані для оновлення. Решта методів має подібну структуру, тому для чистоти коду їх було винесено в абстрактний клас для тестування дій з базою даних. За рахунок цього ми зменшуємо кількість можливих помилок при написанні тестів та уникаємо повтору однакових частин коду. Також витративши декілька хвилин ми можемо написати повноцінний клас для тестування дій з базою даних для будь якого об’єкту.

3.5.5. Метод для тестування реєстрації користувача

Розглянемо тестування функціоналу на прикладі реєстрації. Тестування було проведена на основі метода Black Box. Як можна побачити на Рисунок 3.17. ми не маємо ніяких даних про структуру реалізації реєстрації користувача, ми оперуємо тільки вхідними та вихідними даними. Надсилаємо запит та очікуємо необхідного результату.

```
public class HumanTest extends AbstractTest {

    @Resource HumanResource<Human> humanResource;

    @Override
    public void setResource() { this.resource = humanResource; }

    @Test
    public void checkForCorrectUpdateSpawn() throws Exception {
        assertTrue( condition: true);
    }
```

Рисунок 3.16. Клас для тестування дій з БД

```
    @Test
    public void checkForRegistrationWithWrongEmail() throws Exception {
        for (Human testHuman : testHumans) {
            mockMvc.perform(post( urlTemplate: "/register")
                    .param( name: "name", testHuman.getName())
                    .param( name: "surname", testHuman.getSurname())
                    .param( name: "email", generateRandomString( length: 12))
                    .param( name: "password", testHuman.getPassword()))
                    .andExpect(status().is( status: 422));
        }
    }
```

Рисунок 3.17. Метод для тестування реєстрації

3.5.6. Підсумок тестування

У результаті було створено близько п'ятдесяти тестів, але це всього лиш невелика частина від загальної кількості необхідних тестів. Як показав підключений плагін для тестування, ця кількість тестів покриває всього лиш 31% задіяного функціоналу. На великих проектах запуск тестів може тривати декілька днів. У нашому випадку запуск тестів тривав близько шістнадцяти хвилин.

У результаті був протестований функціонал:

- авторизації користувача;
- реєстрації користувача
- зміна пароля;
- зміна пошти;
- зміна особистих даних;
- додавання книги;
- видалення книги;
- можливість залишити відгук;
- можливість виставити рейтинг книзі;
- можливість завантаження книг;
- пошук книг по назві;
- пошук книг за автором;
- пошук книг за ключовими словами;
- пошук книг за не точним співпадінням слів;

та багато інших тестів. Звіт запуску тестів зображено на Рисунку 3.18.

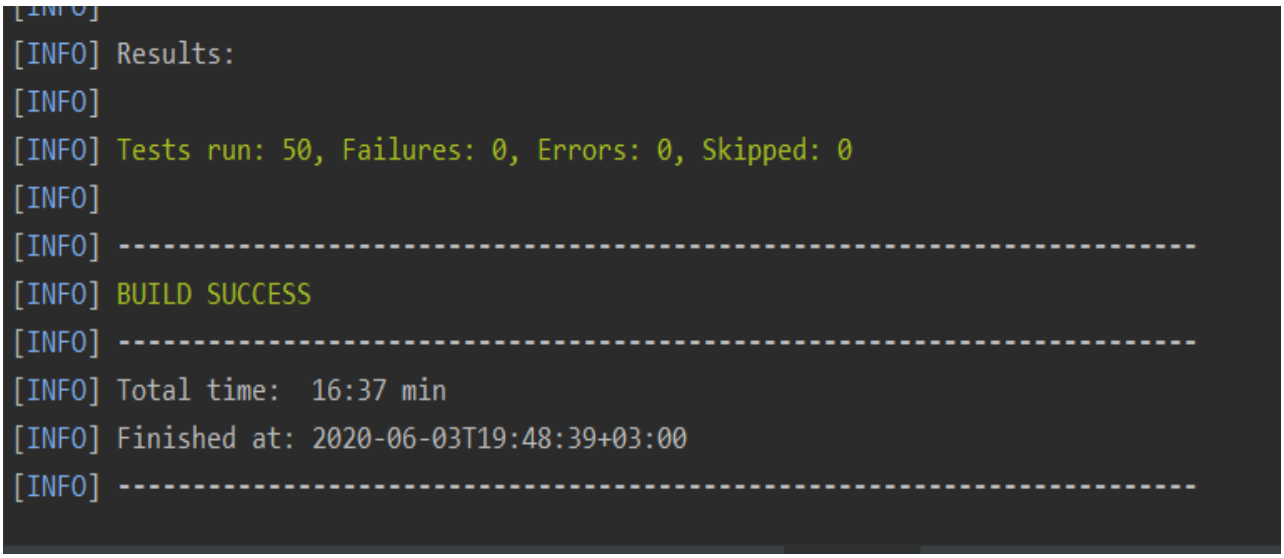


Рисунок 3.18. Результат тестування

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було проведено дослідження і аналіз технологій необхідних для розробки і тестування онлайн бібліотеки.

Було обрано мову програмування, фреймворки, БД та веб-сервіс.

Проаналізовано інструменти для тестування. Розроблено приклад сайту з заповненою базою даних та певним функціоналом проведено тестування відповідних модулів та пошуку книг за запитами, пов’язаними словами.

Налаштований засіб автоматизації роботи з програмними проектами (Maven), для управління та складання програм.

Розроблено примітивний інтерфейс на базі Thymeleaf. Налаштований налаг інтерфейс конфігурації модуля через файл конфігурації Webpack з можливістю повної адаптації синтаксису, що прибирає обмеження використання завантажувача від типу файлу та синтаксису у ньому.

Розроблено програмний код для зв'язку між інтерфейсом та БД та для обробки запитів .

ВИСНОВКИ

Представлений дипломний проект присвячений розробці сайту з підтримкою модульності та текстового пошуку. Головною метою даного проекту є знайти найкраще рішення для проблем з модульністю та пошуком на інтернет сайтах.

У ході виконання роботи було вивчено ринок технологій загальні відомості та проаналізовано стеки розробки, бази даних та інструменти необхідні для розробки. Проаналізовано відгуки користувачів та обрано найкраще рішення.

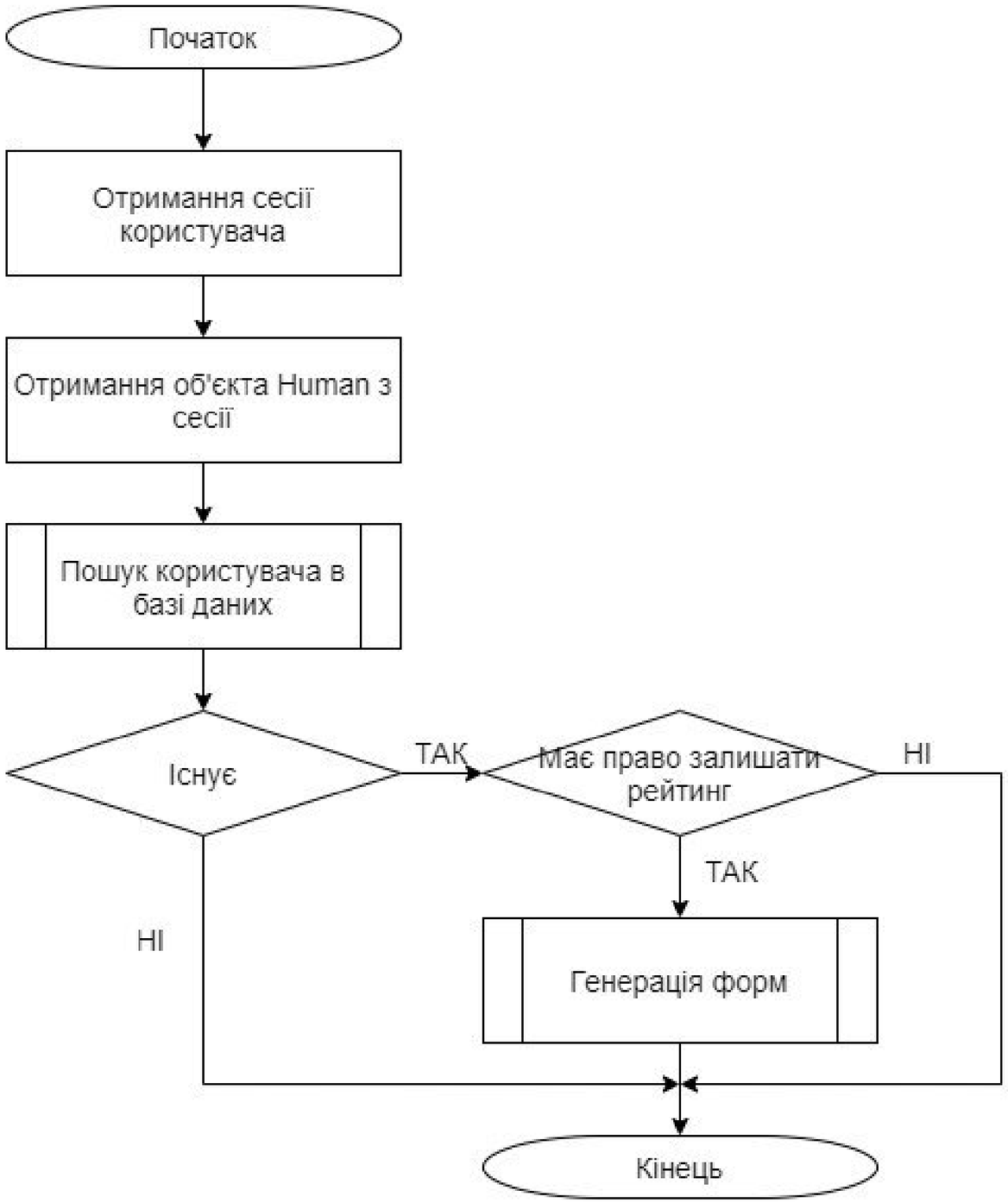
Під час виконання було вивчено обрані технології. Визначено пріоритетні аспекти проблема та алгоритм їх вирішення. Порівняно надані стеки технологій порівнявши їх та виокремивши плюси та мінуси кожної . Розроблено безліч алгоритмів роботи сайту в конкретних ситуаціях враховуючи проблематику. Для вирішення даних проблем було обрано такий стек технологій:

- Менеджер проекту Apache Maven
- Пошукова платформа Apache Solr
- Мова програмування Java
- За основу взято фреймворк Spring

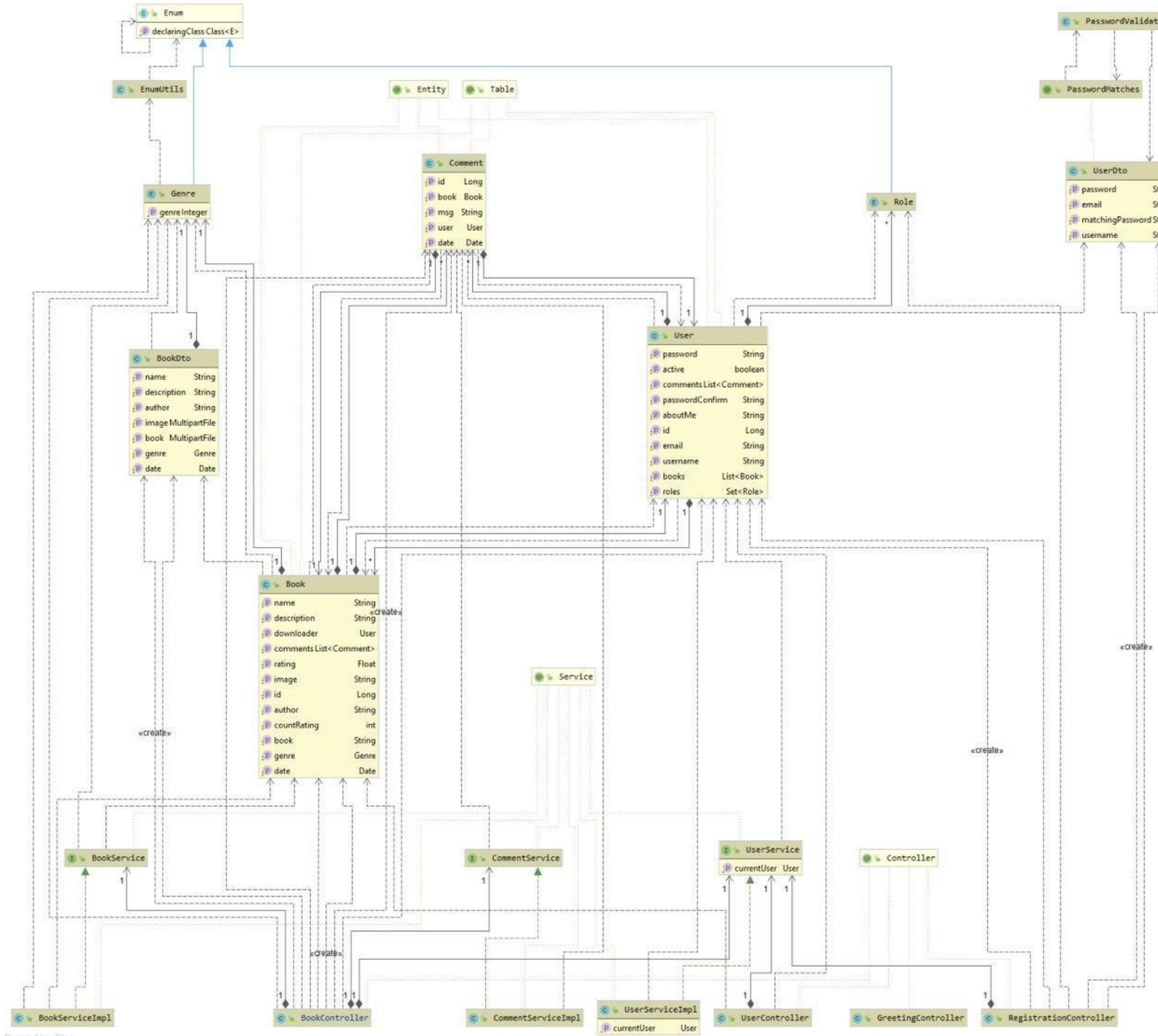
Розроблено онлайн бібліотеку “Spring Book”, як приклад модульного сайту з пошуковою системою.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

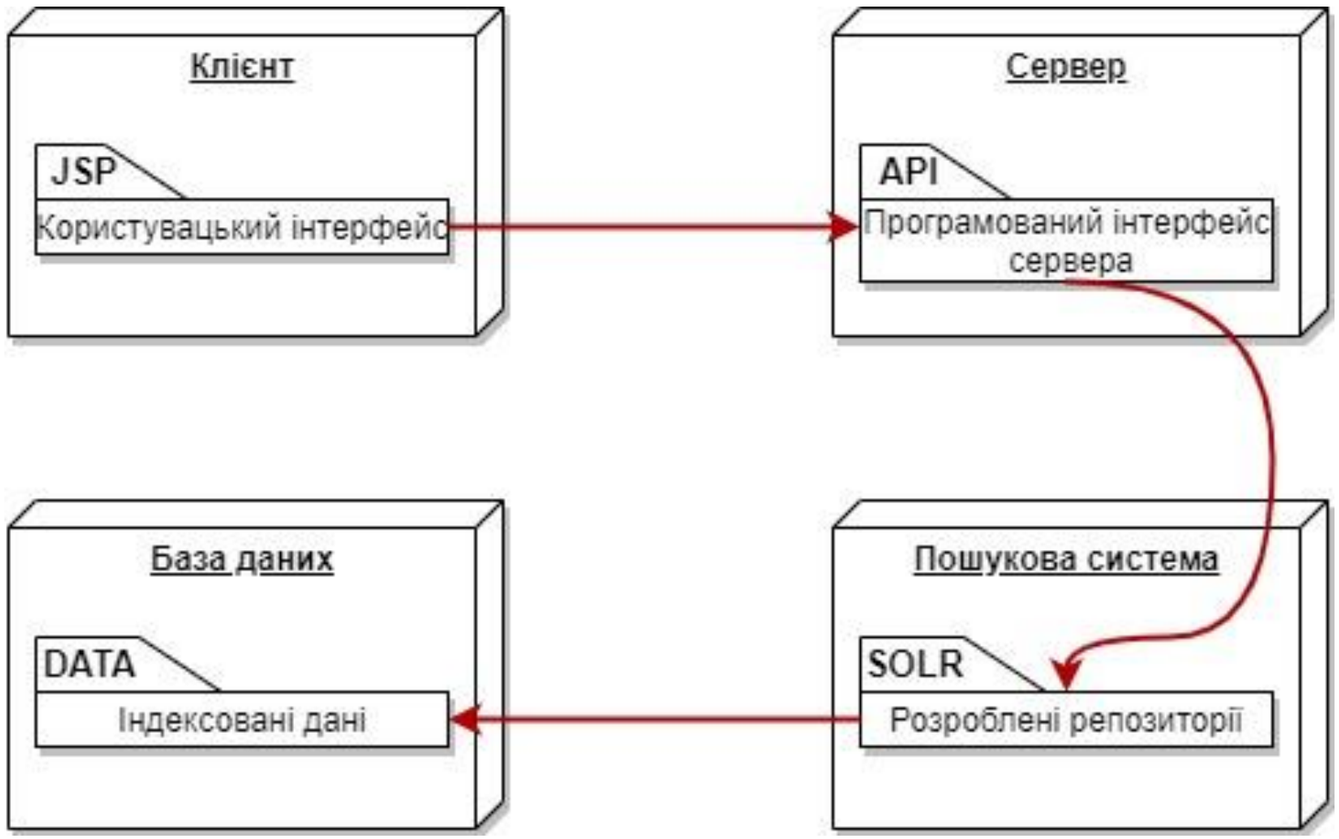
1. Java Platform, Enterprise Edition (Java EE) [Електронний ресурс] – Режим доступу до ресурсу: https://www.theserverside.com/definition/J2EE-Java-2-Platform-Enterprise-Edition
2. Apache Lucene Core [Електронний ресурс] – Режим доступу до ресурсу: https://lucene.apache.org/core/
3. Види тестування ПЗ [Електронний ресурс] – Режим доступу до ресурсу: http://wiki.rosalab.ru/ru/index.php/%D0%92%D0%B8%D0%B4%D1%8B_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%9F%D0%9E
4. Java EE vs Spring [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/18053/.
5. Apache Solr Reference Guide [Електронний ресурс] – Режим доступу до ресурсу: https://lucene.apache.org/solr/guide/8_5/solr-tutorial.html.
6. Top Advantages of Relational Database [Електронний ресурс] – Режим доступу до ресурсу: https://www.educba.com/relational-database-advantages/.
7. Craig W. Spring:Spring in action/Walls, Craig.,2005.
8. Getting started with Mockito [Електронний ресурс] – Режим доступу до ресурсу: https://www.javacodegeeks.com/2015/11/getting-started-with-mockito.html.
9. Schildt H. Java:Java - 8 /Herbert Schildt., 2014.
10. Unit testing with Junit - Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.vogella.com/tutorials/JUnit/article.html.
11. Introduction to Using Thymeleaf in Spring [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/thymeleaf-in-spring-mvc.
12. Discover the Lucene full-text search library[Електронний ресурс] –Режим доступу до ресурсу:http://www.lucenetutorial.com/.



					ІАЛЦ.006328.004 Д1	
Зм.	Арк	№ докум.	Підпи	Дата	Алгоритм відображення книги Схема принципова	Літера / Маса / Масшта
Розроб.		СтельмахА.О				
Перевір.		Антонюк А.І				Аркуш 1 / Аркушів 1
Т. контр.						
Н. контр.		Сімоненко В.П				НТУУ "КПІ" ФІОТ Група ІО-63
Затв.		Стіренко С.Г				



					ІАЛЦ.006328.005 Д2	
					Схема зв'язків класів Схема функціональна	Літера / Маса / Масшта
Зм.	Арк	№ докум.	Підпи	Дата		
Розроб.		СтельмахА.О				Аркуш 1 / Аркушів 1
Перевір.		Антонюк А.І				НТУУ "КПІ" ФІОТ Група ІО-63
Т. контр.						
Н. контр.		Сімоненко В.П				
Затв.		Стіренко С.Г.				



					ІАЛЦ.006328.006 ДЗ	
					Клієнт серверна взаємодія Схема структурна	Літера: Маса: Масшта Аркуш 1 Аркушів 1
Зм.	Арк	№ докум.	Підпи	Дата		
Розроб.		СтельмахА.О				
Перевір.		Антонюк А.І.				
Т. контр.						НТУУ "КПІ" ФІОТ Група ІО-63
Н. контр.		Сімоненко				
Затв.		Стіренко О.Г.				